Introduction

La voix chantée

La voix chantée prend place au cœur de l'expression musicale. On la retrouve dans la plupart des genres musicaux où elle joue souvent un rôle directeur.

> « Le chant est un mode fondamentale d'expression musicale. Il convient particulièrement à l'expression d'une idée spécifique dans la mesure où il est la plupart du temps relié à un texte. Même sans aucun mot, la voix est capable de formuler des déclarations personnelles et identifiables. C'est très probablement le plus subtil et le plus flexible des instruments de musique ce qui explique principalement cette fascination autour de l'art du chant.[1] »

Des machines qui parlent

La diversité presque infinie des sons productibles avec la voix et par conséquent leur complexité ont très tôt intéressé compositeurs et chercheurs. Ainsi, avec l'apparition du phonographe en 1887, l'idée de pouvoir traiter, modifier ou améliorer la voix chantée a fait son apparition. Néanmoins, il a fallu attendre les années soixante avec l'avènement des technologies numériques pour voir l'émergence de techniques de synthèse pour la voix chantée et donc de la notion de vocalité artificielle dans la création musicale.

> « Du phonographe aux voix artificielles entendues dans les œuvres musicales contemporaines, le parcours semble naturel. La voix et la machine y sont à chaque fois convoquées et leur rencontre est la source d'une conception inédite de la vocalité.[2] »

L'idée de pouvoir synthétiser la voix parlée est en revanche largement antérieure à la naissance de l'informatique. En effet, dès l'époque baroque, les facteurs d'orgue tentaient de créer des jeux sur leurs instruments capables de produire des sons de types vocaux. A partir du XVIIe siècle, chercheurs et inventeurs ont travaillé à la création de machines et d'automates capables d'imiter la voix humaine, en particulier la parole. On peut par exemple citer « la machine de Kratzenstein (1777) »[3] qui modélisait un conduit vocal à

---

1 JANDER, Owen, « Singing », *Grove Music Online*, 2010, http://www.oxfordmusiconline.com/subscriber/article/grove/music/25869Singing (en ligne le 04/09/2010), citation traduite de l'anglais : « Singing is a fundamental mode of musical expression. It is especially suited to the expression of specific ideas, since it is almost always linked to a text; even without words, the voice is capable of personal and identifiable utterances. It is arguably the most subtle and flexible of musical instruments, and therein lies much of the fascination of the art of singing. ».

2 BOSSIS, Bruno, *La voix et la machine : La vocalité artificielle dans la musique contemporaine*, Rennes : Presses Universitaire de Rennes, 2005, p. 7.

3 GUILBERT, Jean, *La parole : compréhension et synthèse par les ordinateurs*, Paris : Presses Universitaires de France, 1979, p. 90.

l'aide de résonateurs de formes diverses excités par une anche mise en vibration. Elle permettait de produire des sons voisins des voyelles.

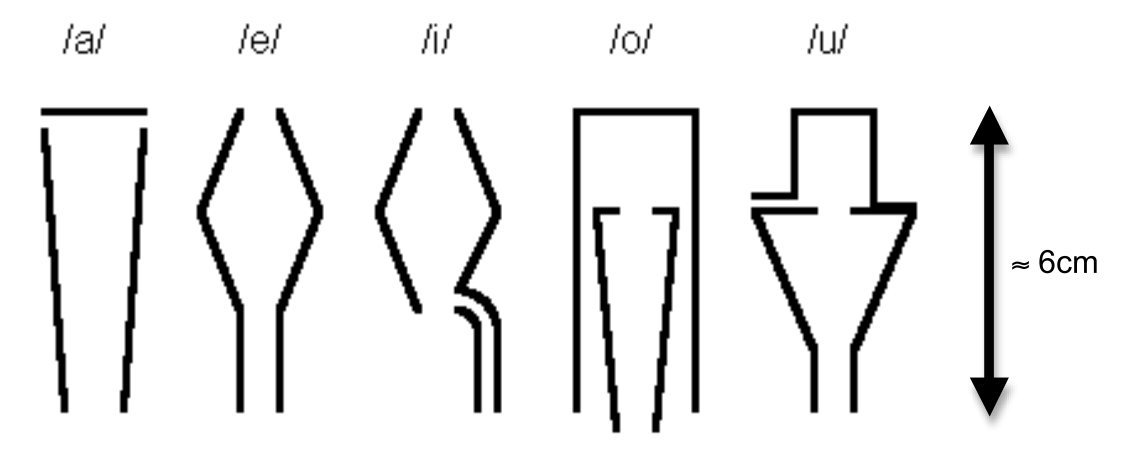


Fig. 1 – Résonateurs de la machine de Kratzenstein pour les différentes voyelles de la langue française.[4] L'air était envoyé dans la partie inférieure de chacun d'entre-eux.

Ce modèle a alors inspiré un grand nombre d'autres projets tel que « la machine de Faber »[5] qui vit le jour en 1835. Celle-ci étaient contrôlée par un clavier et pouvait prononcer des phrases entières.

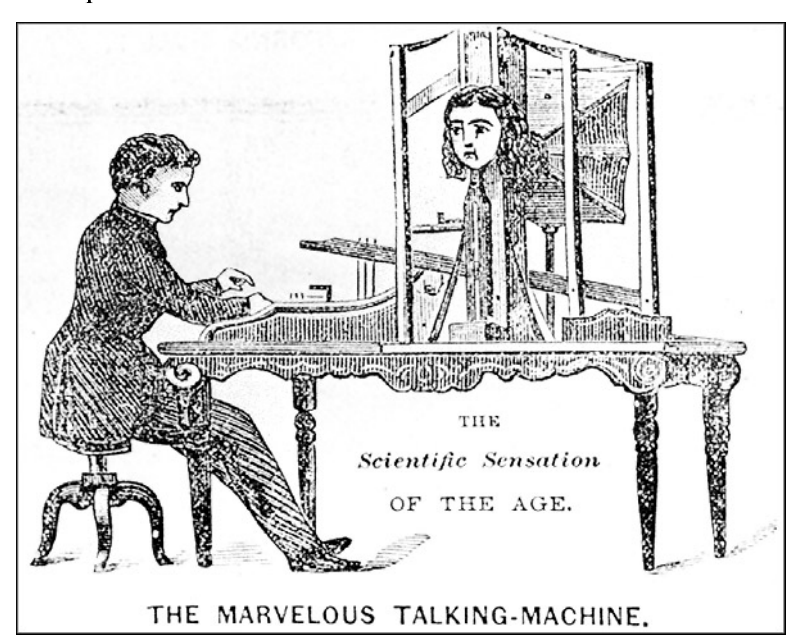


Fig. 2 – La machine de Faber.[6]

Enfin, il est également important de mentionner une machine imaginée par Graham Bell en 1876 : le *Harp telephone* mais qui n'a jamais été construite. Cette dernière constituait le premier système de synthèse de la parole par un procédé électrique. Une rangée

---

4 SCHROEDER, Manfred R., « A Brief History of Synthetic Speech », *Speech Communication*, XIII (1993), n°1 et 2, p. 231-237.

5 LIENARD, Jean-Sylvain, *Les processus de la communication parlée*, Paris : Masson, 1977, p. 107.

6 MILLIKAN, Franck Rives, *Joseph Henry and the Telephone*, http://siarchives.si.edu/history/jhp/joseph23.htm (en ligne le 04/09/2010).

d'anches vibrantes, accordées de telle façon que l'étendue du spectre de la voix soit couvert, devait être placée dans le circuit magnétique d'un électro-aimant.

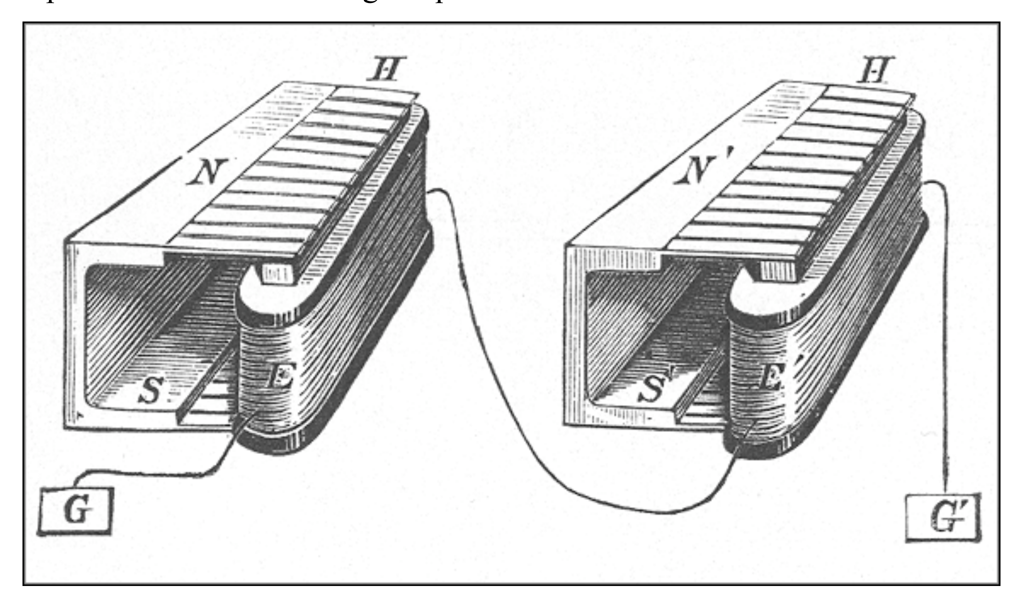


Fig. 3 – Le *Harp Telephone* imaginé par Graham Bell.[7]

Ces modèles primitifs (à l'exception du dernier modèle présenté) de synthèse mécanique que l'on peut qualifier de « modèles physiques » dans la mesure où ils tentaient de reproduire les caractéristiques de l'appareil phonatoire ont été à la base de la création des premiers synthétiseurs numériques de la voix chantée.

Synthétiser la voix chantée aujourd'hui

Les cinquante dernières années ont vu l'émergence d'un important nombre de techniques de synthèse utilisables pour la (re)production de la voix chantée sur ordinateur. Les premières d'entre-elles, basées sur des modèles physiques de tubes acoustiques à une dimension de l'appareil phonatoire offraient des résultats intéressants mais très loins des sons naturels. A partir des années soixante-dix, la demande des compositeurs qui s'intéressaient de plus en plus à la possibilité de pouvoir intégrer des voix de synthèse dans leurs travaux a fortement augmentée.

> « Au milieu du vingtième siècle, la vocalité artificielle commence à prendre place dans la composition musicale […]. Au fur et à mesure de l'apparition de nouveaux modèles et de nouvelles technologies empruntant aux avancées du magnétisme, de l'électronique et de l'informatique, les compositeurs intéressés par des sonorités vocales inouïes, de Pierre Schaeffer et Pierre Henry à Karlheinz Stockhausen,

---

7 PRESCOTT, George Bartlett, *Bell's Electric Speaking Telephone*, New-York : D. Appleton and company, 1884.

> Jonathan Harvey, Philippe Manoury, Kaija Saariaho et bien d'autres, étendent et affinent l'intégration de la vocalité artificielle dans leurs œuvres.[8] »

Les nouvelles techniques numériques qui ont vu le jour au cours de la seconde moitié du vingtième siècle peuvent être classées dans trois grandes catégories : les modèles physiques, les modèles spectraux et les modèles basés sur la concaténation d'échantillons, chacune d'entres elles présentant à la fois divers inconvénients et avantages au niveau de leur implémentation.

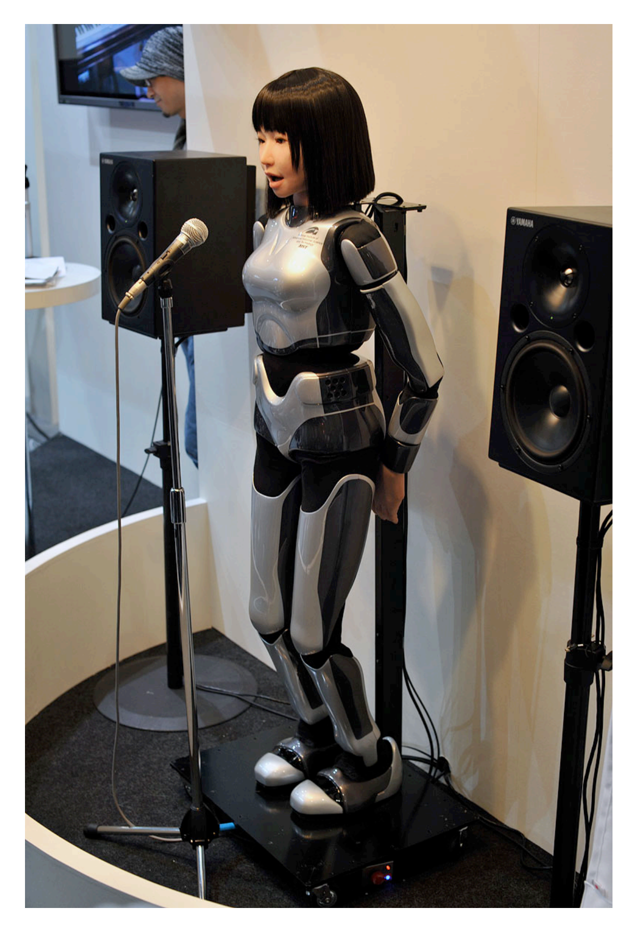


Fig. 4 – Le robot « Miimu » de Yamaha chante grâce à un procédé de synthèse basé sur la concaténation et la modification d'échantillons.[9]

Synthèse de la voix par Fonctions d'Onde Formantique

L'une des techniques rentrant dans cette catégorie a fortement attiré notre attention pour sa simplicité de mise en œuvre et la qualité des résultats obtenus pour la synthèse de sons de type voisés (voyelles). Appelée synthèse par Fonctions d'Onde

8 BOSSIS, Bruno, *La voix et la machine : La vocalité artificielle dans la musique contemporaine*, Rennes : Presses Universitaire de Rennes, 2005, p. 285-286.

9 http://tokyobling.wordpress.com/tag/music/ Blog de Tokyobling (en ligne le 04/09/2010).

Formantique (ou FOF), elle a été développée à l'IRCAM[10] au début des années quatre-vingt par l'équipe de Xavier Rodet. Elle peut être classée dans deux des catégories mentionnées précédemment dans la mesure où elle est basée sur un modèle à la fois physique et spectral. Cette technique a connu un essor important au cours des années quatre-vingt et a été utilisée par un certain nombre de compositeurs tant pour la synthèse de son vocaux que pour la synthèse d'autre type de sons. En effet, elle s'est révélée être particulièrement efficace dans la production de sons instrumentaux percussifs suivis d'une résonnance, comme ceux produits par les xylophones, les instruments à cordes pincées et frappées, etc.

Le programme CHANT

La première implémentation de la synthèse par Fonctions d'Onde Formantique a été faite à travers le programme CHANT développé à l'IRCAM par Xavier Rodet et Pierre-François Baisnée. CHANT est un synthétiseur par règle permettant la synthèse de voyelles avec un nombre réduit de paramètres. « Chant est le synthétiseur le plus simple à manier »[11]. Il est également important de noter qu'un générateur de bruit filtré permettait la synthèse de sons de type fricatifs.

Des exemples de démonstration très impressionant du programme ont été faits à cette période, on peut par exemple citer la synthèse de *L'air de la reine de la nuit* de *La Flute enchantée* de Mozart[12] [13] devenu aujourd'hui célèbre.

CHANT a été adapté pour fonctionner sur différentes plateformes jusqu'en 1992 avec la bibliothèque *Chant-PW* dans PatchWork. A partir de 1995, le programme PatchWork laissa progressivement place à OpenMusic. *Chant-PW* n'a jamais été portée dans ce programme mettant alors fin à la dernière version complète de CHANT.

Après cette date, la synthèse FOF a connu un désintérêt progressif et a été peu à peu oubliée. Bien que le synthétiseur CHANT subsiste encore dans des versions simplifiées à travers certains programmes comme Max/MSP ou Diphone Studio, on est très loin des versions antérieures du programme qui offraient un bien plus grand nombre de possibilités.

---

10 Institut de Recherche et Coordination Acoustique/Musique

11 BAISNEE, Pierre-François, *CHANT manual*, Document Ircam : Paris, 1985, p. 1.

12 Disponible sur le cd à cd/audio/reine-IRCAM.aiff.

13 Interview télévisée de Max Mathews sur la musique assistée par ordinateur, vidéo disponible sur le cd dans le fichier « Mathews-reine.m4v » à cd/vidéo et sur le site YouTube à l'adresse suivante : http://www.youtube.com/watch?v=_15ZQL82P4M (en ligne le 04/09/10).

Les solutions offertes par le programme CHANT pour la synthèse de la voix mais aussi pour la synthèse de sons en général sont toujours d'actualité. L'étude et la comparaison des différentes techniques actuellement disponibles dans ce domaine que nous avons faite montre que les sons produits par la synthèse FOF pour l'imitation de sons de type voisés font encore partie des meilleurs résultats disponibles.

PatchWork et OpenMusic sont tous deux « des environnements de développement visuel pour la création d'applications de composition musicale assistée par ordinateur »[14]. Les traitements des informations s'y font en temps différé, tout comme dans le cas de CHANT.

Il sera donc montré au cours de notre recherche que ces deux programmes sont assez similaires notamment au niveau de leur architecture. On peut donc imaginer que le portage de la bibliothèque *Chant-PW* vers OpenMusic peut être une opération relativement simple à exécuter. Ceci rendrait à nouveau accessible le programme CHANT ainsi que toutes ses fonctions.

La synthèse de la voix chantée par Fonctions d'Onde Formantique – techniques, outils existants, exemple d'implémentation et utilisation : définition du sujet

L'appareil phonatoire chez l'homme est la source de la production de la voix chantée. Il est composé d'un ensemble d'organes formant un système complexe permettant la phonation. Afin de comprendre les modèles de synthèse visant à reproduire des sons de types vocaux, il est nécessaire de posséder une bonne connaissance des différents éléments de cet appareil phonatoire. Ainsi, dans un premier temps, une brève description de ce dernier sera faite d'un point de vue anatomique. Il sera également question de présenter les différentes interactions entre les éléments qui le constituent et de montrer le rôle de chaque organe dans la phonation.

Les recherches dans le domaine de la synthèse de la voix chantée ont permis la création d'un nombre conséquent de techniques.

---

14 http://recherche.ircam.fr/equipes/repmus/OpenMusic/index.html Site officiel du programme OpenMusic (en ligne le 04/09/2010).

> « A partir du début des années soixante, la voix chantée a été synthétisée par l'ordinateur. Depuis ces toutes premières expériences, la qualité musicale et le naturel des voix chantées de synthèse a largement été amélioré, […].[15] »

La plupart de ces techniques seront présentées afin d'établir un « état des lieu » du domaine et de les comparer à la synthèse par Fonctions d'Onde Formantique. Cette dernière fera l'objet d'un développement complet dans lequel une description détaillée de son fonctionnement sera faite. Les différents procédés de traitement numérique du signal (DSP[16]) mis en jeu feront l'objet d'une implémentation dans les programmes MATLAB et CSOUND.

L'une des principales implémentations de la synthèse par FOF est le programme CHANT. Un historique de son évolution sera dressé et son fonctionnement sera décrit précisément. Il sera aussi question de présenter les différents programmes faisant encore appel à lui de façon direct où indirect.

Nous proposerons une solution non exhaustive de ré-implémentation de CHANT dans le programme OpenMusic interfacé avec le programme CSOUND qui effectuera l'ensemble des taches de traitement numérique du signal relatives à la synthèse par Fonction d'Onde Formantique. Les codes LISP et CSOUND mis en jeu feront l'objet d'une analyse détaillée.

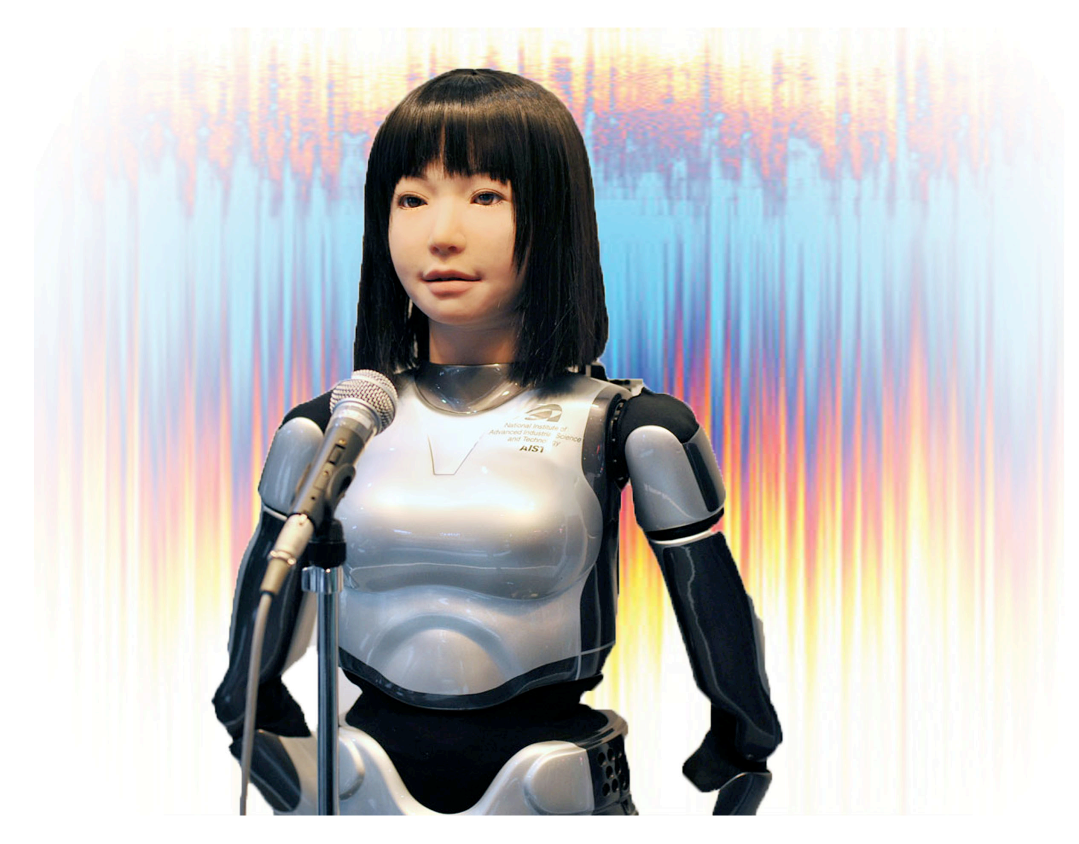
Enfin, le système créé sera testé à travers le célèbre exemple de la synthèse de *L'air de la reine de la nuit* de *La flute enchantée* de Mozart. Nous tenterons d'obtenir des résultats similaires à ceux de l'équipe de Xavier Rodet (1984).

---

15 RODET, Xavier, *Synthesis of the Singing Voice : acte du colloque Benelux Workshop on Model Based Processing and Coding of Audio, Leuven (Belgique), 15/11/2002*, Paris : IRCAM, 2003, p. 99, citation traduite de l'anglais : « As soon as the beginning of the 60s, the singing voice have been synthesized by computer. Since thèse first experiments, the musical and natural quality of singing voice synthesis has largely improved, […]. ».

16 Digital Signal Processing.

I. Synthèse de la voix humaine par ordinateur



Le robot « Miimu » de Yamaha chante grâce à un procédé de synthèse basé sur la concaténation et la modification d'échantillons (cf. figure 4).

A. L'appareil phonatoire humain

Pour comprendre les caractéristiques sonores d'un instrument de musique, il est indispensable d'étudier son organologie. La voix humaine, même si elle a bien d'autres fonctions, peut-être utilisée à des fins musicales. On ne parle alors plus au niveau de sa constitution d'organologie mais d'anatomie.

Dans ce chapitre, nous allons tenter de mettre en relation les caractéristiques anatomiques de l'appareil phonatoire et les propriétés du son qui en résultent.

a) Anatomie de l'appareil phonatoire[17]

L'appareil phonatoire peut être divisé en trois parties différentes : le système respiratoire, les cordes vocales et le système articulatoire.

Le système respiratoire est principalement constitué des poumons et est relié aux deux autres parties de l'appareil phonatoire par la trachée. Les poumons sont eux-mêmes formés de petites cavités appelées alvéoles reliées à la trachée par les bronches. Nous verrons par la suite qu'ils jouent principalement un rôle de compresseur d'air dans la production de la voix.

Les cordes vocales, qui par ailleurs ont plutôt la forme de « plis », sont aussi appelées « plis vocaux ». Elles sont constituées de tissus musculaires recouverts de muqueuses plus ou moins visqueuses. La longueur moyenne des cordes vocales dépend de l'âge et du sexe de l'individu. Ainsi, elles mesurent approximativement trois millimètres chez le nourrisson, neuf à treize millimètres chez la femme et quinze à vingt millimètres chez l'homme adulte. Plus les cordes vocales sont longues, plus elles permettent la production de sons graves.

Comme on peut le voir sur la figure 6, les cordes vocales ont pour point d'origine la thyroïde et sont directement rattachées aux cartilages aryténoïdes par des ligaments. Précisons que la partie externe de la thyroïde est parfois visible chez certains sujets males à l'âge adulte et est plus communément appelée la « pomme d'Adam ». L'espace entre les cordes vocales est appelé glotte.

---

17 SUNDBERG, Johan, *The Science of the Singing Voice*, Dekalb (Illinois) : Northern Illinois University Press, 1987, p. 6-8.

Les cartilages aryténoïdes sous l'action des muscles crico-arytenoïdiens et inter-arytenoïdiens jouent un rôle primordial dans la production de la voix. En effet, ils ouvrent ou ferment la glotte en mettant en contact les deux cordes vocales qui émettent alors un son (figure 7). L'action de fermeture de la glotte est appelée adduction et l'action d'ouverture abduction. De ce fait, ces deux mouvements permettent ou non l'émission de son par les cordes vocales. Les sons vocaux tels que les voyelles qui sont produits lors d'une adduction sont dits « voisés ». A l'inverse, les sons produits lors d'une abduction sont dits « non-voisés ». Par exemple, lorsque nous prononçons le mot « casser », les cordes vocales sont adductées jusqu'à l'arrivée sur le son /s:/ où elles deviennent donc abductées.

Les cordes vocales ne sont pas le seul organe apte à produire du son dans l'appareil phonatoire. En effet, il est possible de distinguer une autre paire de membranes se trouvant quelques millimètres au dessus des cordes vocales appelées « bandes ventriculaires ».

La courte section de l'appareil phonatoire décrite précédemment est appelée le larynx. De façon schématique, il s'étend de la glotte jusqu'à la partie inférieure de l'épiglotte et mesure entre un et deux centimètres. Il est directement inséré dans un tube plus long et plus large appelé pharynx.

C'est à la partie inférieure du larynx que l'on trouve le cartilage cricoïde qui permet l'ouverture ou le fermeture de la trachée et de l'œsophage. Ainsi, lors de la phonation, l'œsophage est fermé et la trachée ouverte, lors de l'ingestion d'aliments, l'œsophage est ouvert et la trachée fermée.

Le système articulatoire constitue la partie terminale de l'appareil phonatoire. Il s'étend de la zone supérieure du larynx jusqu'aux lèvres. Il est constitué dans sa partie inférieure de la langue, des dents et des lèvres inférieures. On trouve dans sa partie supérieure le palais, la cavité nasale, les sinus, les dents et les lèvres supérieures. La cavité nasale est reliée à la partie antérieure de la bouche par un conduit qui peut-être ouvert ou fermé par la luette (ouvert dans le cas de la respiration, fermé dans le cas de l'ingestion). Toutes les données anatomiques présentées précédemment sont schématisées dans la figure 5.

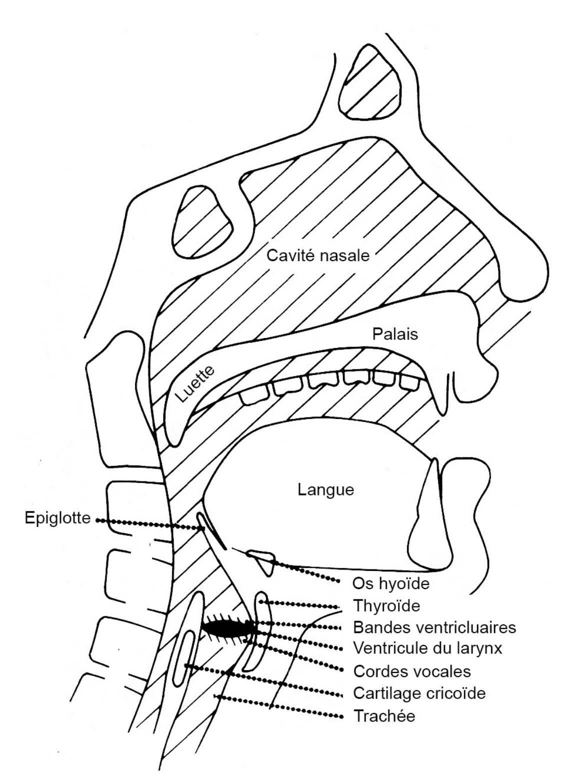


Fig. 5 – Vue d'ensemble de l'appareil phonatoire chez l'Homme.[18]

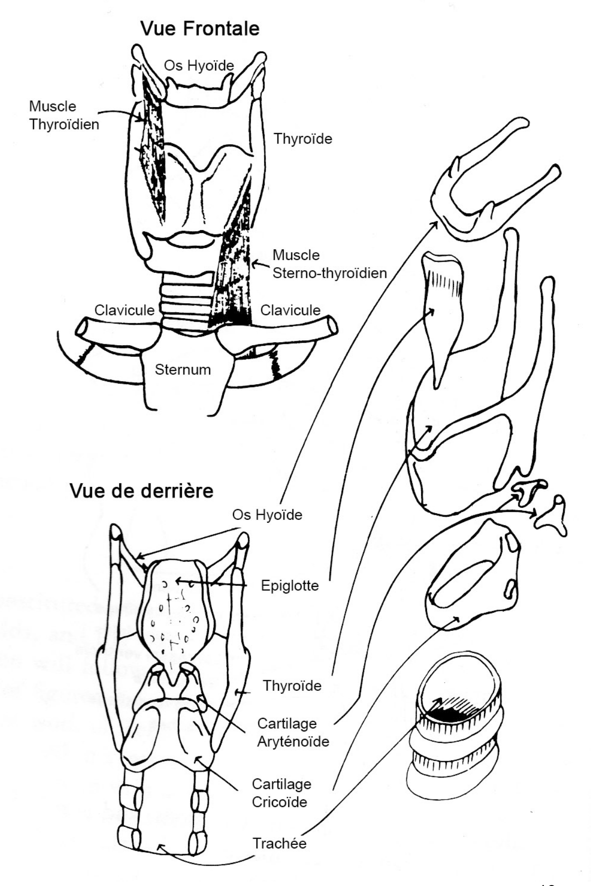


Fig. 6 – Les différents cartilages du larynx.[19]

18 *Ibid.*, p. 8.

19 *Ibid.*, p. 8.

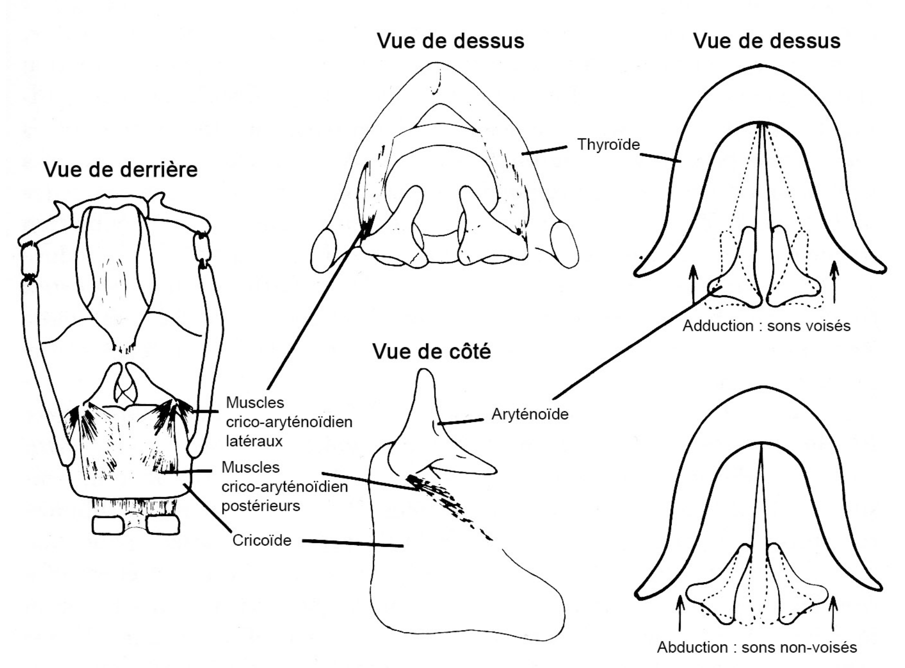


Fig. 7 – Adduction et abduction des cordes vocales.[20]

La description de l'appareil phonatoire qui vient d'être faite reste assez schématique. Les organes qui le constituent ont une taille très réduite et sont particulièrement complexes. La configuration générale de ce système dépend d'un nombre très important de paramètres qui influent sur la nature des sons produits.

b) Mécanismes de production de la voix[21]

Dans le chapitre précédent, les différents éléments de l'appareil phonatoire ont été décrits d'un point de vue anatomique. Dans cette partie, le rôle et le fonctionnement du système respiratoire et vocal dans la production de la voix vont être expliqués.

o Fonctionnement global du système phonatoire

D'un point de vue physiologique, chacun des systèmes présentés précédemment joue un rôle important. De manière schématique, les poumons forment un compresseur créant un flux d'air envoyé aux cordes vocales par l'intermédiaire de la trachée.

---

20 *Ibid.*, p. 8.

21 *Ibid.*, p. 9-18.

Le son émis par les cordes vocales qui rentrent alors en vibration sous l'influence des cartilages aryténoïdes est appelé source vocale. Il est ensuite acheminé jusqu'au système articulatoire qui joue un rôle de résonateur modifiant la nature du son produit. Ces mécanismes complexes seront décrits plus en détail dans la partie (c) de ce chapitre.

Techniquement parlant, le système vocal général peut donc être résumé de la façon suivante : un compresseur (les poumons) alimentant un oscillateur (les cordes vocales) filtré par des filtres résonants (la cavité buccale) comme le résume le schéma suivant :

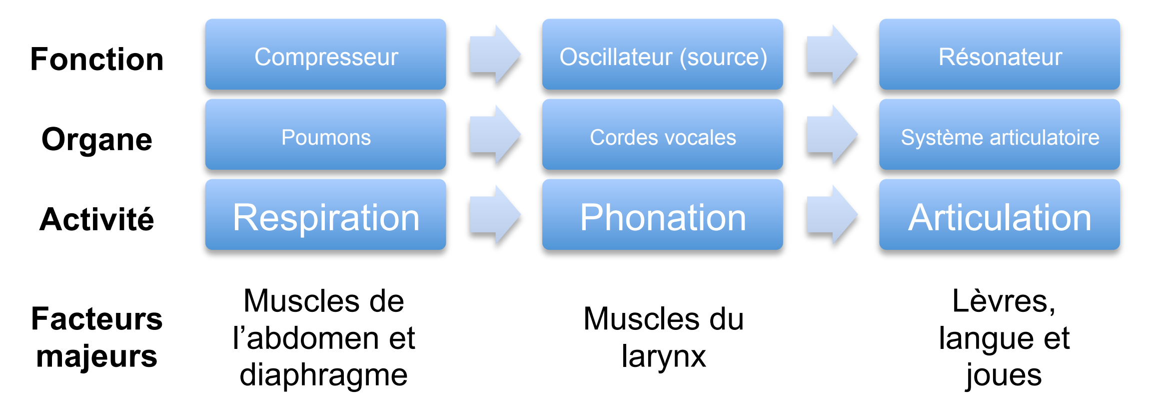


Fig. 8 – Diagramme représentant les trois fonctions principales du système phonatoire sous différents points de vue – Inspiré du diagramme de Johan Sundberg[22]

○ Création du signal de la source vocale

Un son est un signal acoustique variant au cours du temps composé de variations minuscules et rapides de la pression de l'air. Dans le cas présent, ces variations sont créées par les cordes vocales. Lorsque le flux d'air généré par les poumons passe entre les deux cordes vocales qui sont alors en position adductée, elles s'ouvrent et se ferment de façon très rapide et alternative découpant le flux d'air en une succession d'impulsions formant un signal acoustique (figures 9 et 10). Ce cycle étant répété *n* fois par secondes, *n* détermine la fréquence f_0 exprimée en Hertz de la fondamentale du son généré. Ainsi, lorsqu'un ténor chante un « LA » à une fréquence de 220Hz, les cordes vocales s'ouvrent et se ferment 220 fois par seconde. On peut noter que la notion d'impulsion est très importante dans la compréhension de la méthode de synthèse par Fonctions d'Onde Formantique qui sera présentée en détail dans le chapitre II.

---

22 *Ibid.*, p. 10.

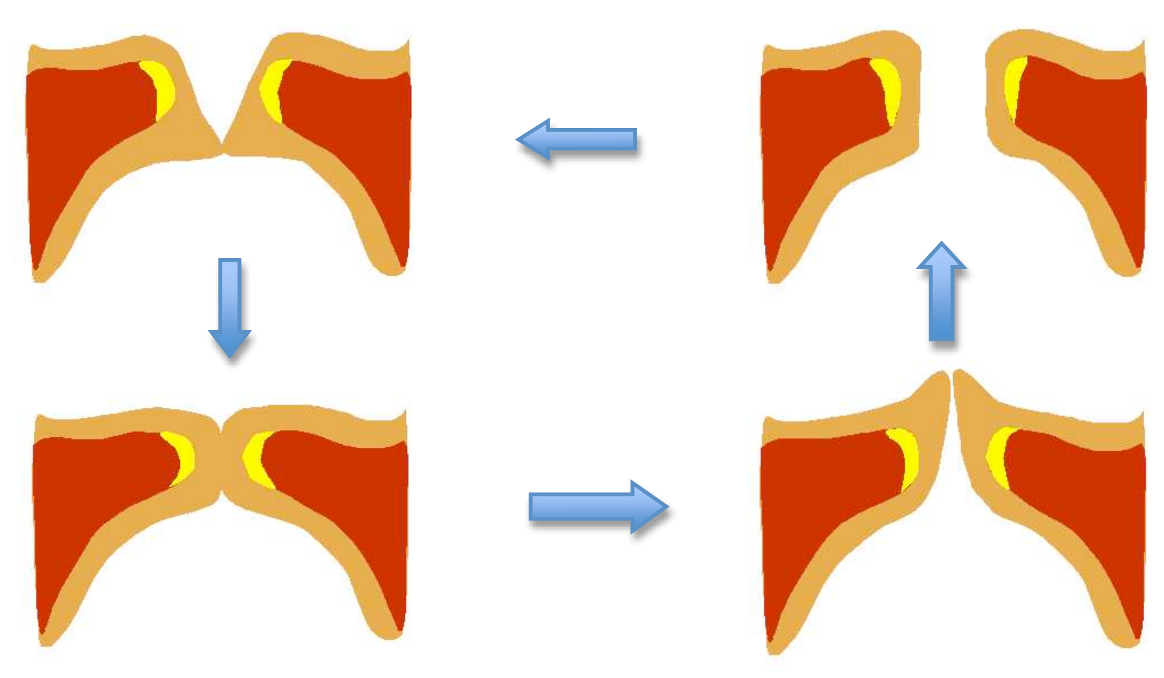


Fig. 9 – Cycle d'ouverture et de fermeture des cordes vocales lors de la phonation.[23]

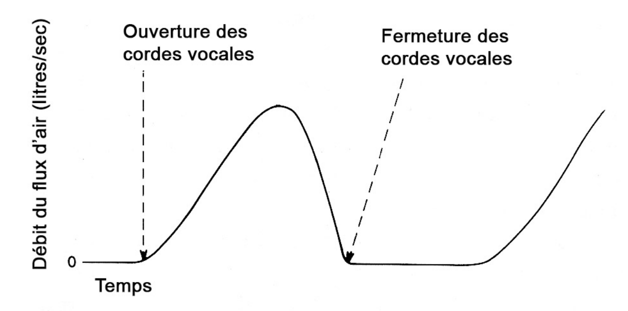


Fig. 10 – Représentation schématique d'une impulsion créée par l'ouverture puis la fermeture des cordes vocales.[24]

- **Contrôle de la fréquence de vibration des cordes vocales**

Dans tout langage et pour la voix chantée, il est primordial d'être en mesure de modifier sa fréquence de phonation. Pour ceci, deux paramètres doivent être pris en compte : la pression du flux d'air généré par les poumons et la tension des cordes vocales. Plus la pression du flux d'air est importante, plus la fréquence de phonation aura une valeur élevée et plus l'amplitude du son généré sera forte. Il faut toutefois préciser que l'influence de la pression du flux d'air sur la fréquence du son émis reste assez minime. La longueur et donc la

---

23 http://fr.wikipedia.org/wiki/Fichier:Vocal_fold_animated.gif (en ligne le 04/09/2010).

24 SUNDBERG, Johan, *op. cit.*, p. 11.

tension des cordes vocales est donc le paramètre le plus important dans la détermination de leur fréquence de vibration.

En effet, plus les cordes vocales sont longues et fines plus elles produisent un son aigu et vice-versa. La longueur des cordes vocales est contrôlée par la distance entre les points d'origines de leurs extrémités : la tyroïde et le cartilage aryténoïde sous l'influence des muscles crico-thyroïdiens.

Enfin, il est important de noter que lorsqu'un chanteur exécute un crescendo en se maintenant sur la même note, il ajuste progressivement et de façon inconsciente la longueur de ses cordes vocales afin d'éviter que sa fréquence de phonation augmente.

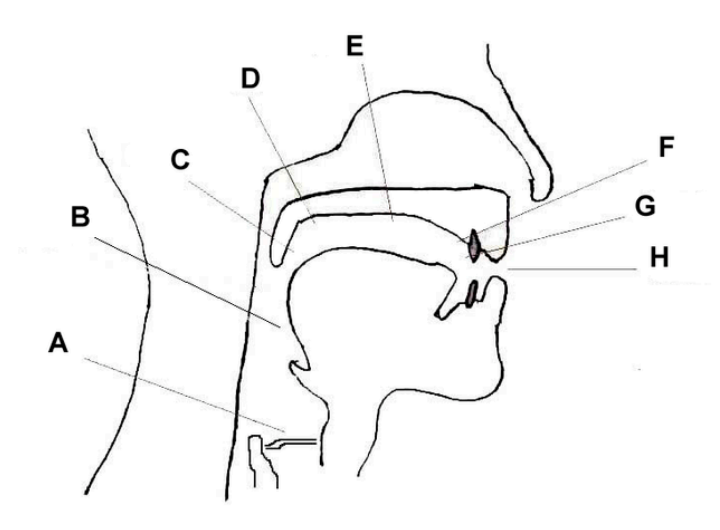
- **Les autres générateurs de son du système phonatoire**

Comme cela a été énoncé précédemment, les cordes vocales ne permettent de générer que des sons « voisés » périodiques comme les voyelles. Or la plupart des langages chez l'Homme comportent également des sons apériodiques tels que les consonnes. De ce fait, d'autres parties du système vocal peuvent jouer un rôle de source sonore.

Lorsque le flux d'air provenant des poumons est propulsé au travers de petits espaces très restreints où il est comprimé puis libéré de façon subite, il devient turbulent et on observe alors la création d'un signal acoustique de type bruité. A la différence du son produit par les cordes vocales, celui-ci est apériodique. D'un point de vue phonétique, ce type de son est assimilé à ceux des consonnes.

Il existe donc deux grand type de consonnes : les consonnes occlusives qui sont produites lors d'une ouverture rapide de la bouche et les consonnes continues résultant du passage du flux d'air provenant des poumons dans un espace restreint (sons fricatifs, latéraux et vibrants).

Les dix-huit consonnes de la langue française peuvent êtres classées de la manière suivante :



	Points d'articulation							
Mode d'articulation	**A** : consonnes laryngales	**B** : consonnes pharyngales	**C** : consonnes uvulaires	**D** : consonnes vélaires	**E** : consonnes palatales	**F** : consonnes alvéolaires	**G** : consonnes dentales	**H** : consonnes labiales
Occlusives			/q:/ et /ɢ:/	/g:/ et /k:/	/c:/ et /ɟ:/	/d:/ et /t:/	/t̪:/ et /d̪:/	/b:/ et /p:/
Fricatives	/ħ:/, /ʕ:/, /ʜ:/ et /ʢ:/	/h:/ et /ɦ:/	/ʁ:/	/x:/ et /ɣ:/	/ç:/ et /ʝ:/	/z:/, /s:/, /ʒ:/ et /ʃ:/	/θ:/ et /ð:/	/v:/ et /f:/
Nasales			/ɴ:/	/ŋ:/	/ɲ:/	/n:/	/n̪:/	/m:/
Spirantes latérales				/ʟ:/	/ʎ:/	/l̩:/	/l̪:/	

Fig. 11 – Classement des consonnes de la langue française (en orange) et de quelques autres langues européennes (en bleu) en fonction de leurs points et de leurs modes d'articulation.[25] [26] [27]

Afin de mieux comprendre le tableau donné précédemment, il est nécessaire de donner quelques exemples concrets :

- Si l'on prononce le son /f:/ que l'on retrouve au début du mot fantôme, une minuscule fente est créée entre la lèvre inférieure et les incisives supérieures.
- Un autre générateur de bruit peut aussi être créé entre la partie antérieure de la langue et la luette. Il peut servir à générer des sons comme /ʁ:/ que l'on retrouve au début du mot « ronronner ».
- Des sons de type apériodique peuvent aussi être produits au niveau des cordes vocales en position abductée comme dans le cas de chuchotements. Elles sont alors tellement tendues qu'elles n'entrent pas en vibration mais l'espace créé est si étroit qu'il engendre des turbulences dans le flux d'air en provenance des poumons générant ainsi du bruit.

---

25 http://fr.wikipedia.org/wiki/wiki/Fichier:Places_of_articulation.png (en ligne le 06/06/2010).

26 BOITE, René, *Traitement de la parole*, Lausanne : Presses polytechniques et universitaires romandes, 2000, p.15-16.

27 http://weston.ruter.net/projects/ipa-chart/view/ Site officiel de l'alphabet phonétique international (en ligne le 04/09/2010).

Dans cette partie, la plupart des sources sonores du système phonatoire ont été présentées. Elles permettent la création d'une importante palette de sons différents qui constituent en partie la phonétique d'un langage donné. Pour diversifier encore plus ces sons et en particulier ceux produits par les cordes vocales, l'Homme utilise sa cavité buccale comme résonateur. La partie suivante est consacrée à l'étude du système articulatoire de l'appareil phonatoire.

c) De la source sonore au son perçu : l'articulation[28]

Dans le cas où plusieurs individus émettent une même note, il est reconnu que la voix de chacun d'entre eux est différente. Or, les recherches dans ce domaine montrent que le son produit par les cordes vocales ne diffère que peu d'un individu à un autre. Ainsi, les caractéristiques sonores qui nous sont propres ne proviennent pas seulement de la nature de nos cordes vocales mais aussi de la configuration de notre cavité buccale.

- **Les formants**

Les théories de l'acoustique montrent que les sons périodiques peuvent être décomposé en un ensemble de sons sinusoïdaux appelés partiels ayant chacun une fréquence différente. Pour un son périodique donné, le partiel dont les fréquences de tous les autres sont multiples est appelé la fondamentale et se note f_0. Celle-ci détermine la note perçue lors de l'émission d'un son. Les partiels multiple de f_0 sont dits « harmoniques ».

Les cordes vocales émettent un son très riche du point de vue spectral. Ce son est acheminé jusqu'à la cavité buccale qui va le modifier en amplifiant certains de ces harmoniques en les faisant « résonner », d'où le terme « résonateur » utilisé précédemment. Les bandes de fréquences amplifiées sont appelées formants.

Le concept de formant est primordial dans le système vocal dans la mesure où ils déterminent en grande partie le timbre de la voix d'un individu donné ainsi que le type de voyelle chantée.

---

28 *Ibid.*, p. 19-24.

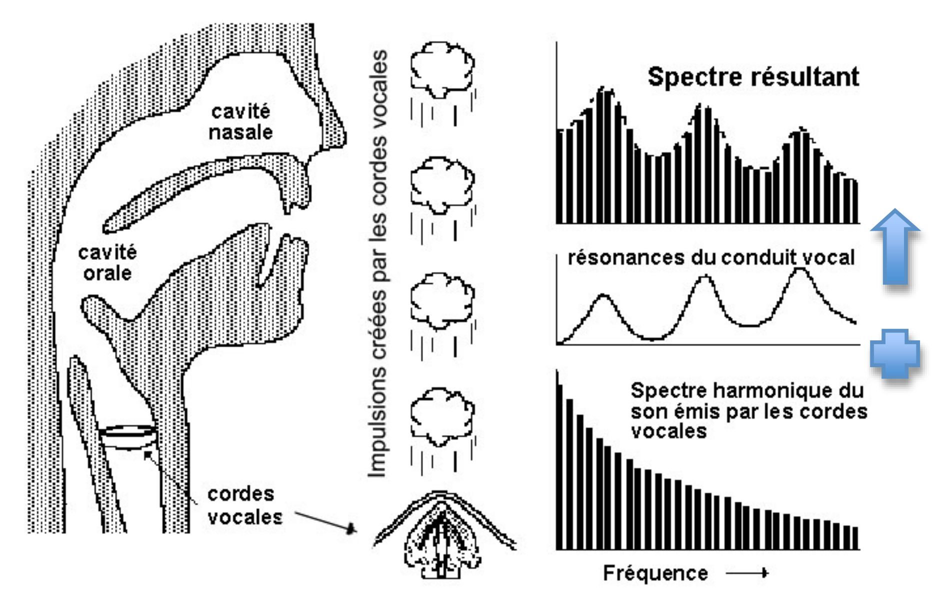


Fig. 12 – Filtrage du son émis par les cordes vocales dans le système articulatoire.[29]

- **Les articulateurs**

Il est maintenant important d'introduire la notion d'articulateur. En effet, chaque organe constituant le système articulatoire joue un rôle dans la modification du son émis par les cordes vocales. Ainsi, les articulateurs sont la mâchoire, la langue, la luette et le larynx.

Les fréquences des formants dépendent donc principalement de la forme et de la taille de la cavité buccale qui peuvent être modifiées par les articulateurs. La taille est définie par la distance entre les lèvres et la glotte. Ainsi, la cavité buccale peut-être schématisée par une courbe sur laquelle l'axe des abscisses représente la distance depuis les lèvres jusqu'à la glotte et l'axe des ordonnées la taille de l'espace présent à un endroit donné :

29 BELANGER, *Olivier, Analyse, synthèse et traitement des sons*, http://cours.musique.umontreal.ca/mus1321/Notes_de_cours/Csound_06_Voix.html (en ligne le 09/04/2010).

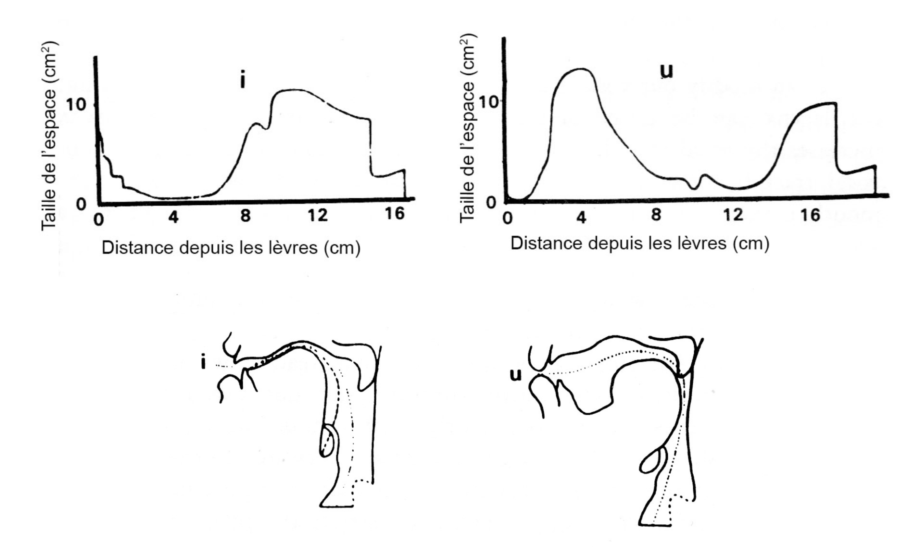


Fig. 13 – Représentation graphique du système articulatoire pour les sons /i:/ et /u:/.[30]

La taille de la cavité buccale peut-être modifiée de deux façons différentes. Il est tout d'abord possible d'allonger la bouche en poussant les lèvres vers l'extérieur. Le larynx peut également être déplacé. Ces deux actions se font de façon très naturelle lorsque l'on parle, de plus, la position du larynx dépend aussi fortement de la fréquence du son produit.

Le principal effet de l'allongement de la cavité buccale est l'abaissement de la fréquence de tous les formants. On peut donner en exemple le son /i:/ qui devient /y:/ lorsque les lèvres sont poussées vers l'extérieur.

La forme globale de la cavité buccale joue aussi un rôle important dans la détermination du timbre de la voix et des voyelles prononcées. Elle peut-être modifiée d'un très grand nombre de façons. La langue tout d'abord, peut être aplatie ou arrondie. Elle peut aussi être enflée vers l'avant (son /i:/) ou vers l'arrière (son /u:/).

La luette peut être relevée ou détendue. Dans ce dernier cas, le son est envoyé vers la cavité nasale qui comme la cavité buccale fait office de résonateur. Les sons produits dans de telles conditions sont dits « nasaux ».

En plus d'être déplacé, le larynx peut aussi facilement changer de forme en particulier grâce à la grande mobilité des cartilages aryténoïdes.

- **Le rôle de chaque formant**

30 SUNDBERG, Johan, *op. cit.*, p. 21.

Bien que le nombre de formants générés par le système articulatoire soit très important, les cinq premiers d'entre eux suffisent à définir très précisément les voyelles prononcées et le type de voix perçu.

Le premier formant est particulièrement sensible au taux d'ouverture de la mâchoire. En effet, plus celle-ci est ouverte, plus il aura une fréquence élevée, comme pour le son /α:/ présent dans le mot « papa » par exemple.

La fréquence du second formant change principalement en fonction de la forme de la langue. Dans le cas où celle-ci se rétracte vers l'arrière de la cavité buccale, la fréquence prend des valeurs très élevées comme c'est le cas pour /u:/ que l'on retrouve par exemple dans le mot « douche ». A l'inverse, si la langue est poussée vers l'avant de la bouche se rapprochant alors du palais, le second formant est abaissé.

De manière générale, ces deux premiers formants permettent de déterminer la plupart des voyelles comme cela est montré dans la figure 14.

A titre indicatif, le premier formant varie généralement entre 250 et 1000 Hz chez l'homme adulte, le second entre 600 et 2500 Hz et le troisième entre 1700 et 3500 Hz. Les autres formants sont beaucoup moins mobiles que les trois premiers et leur fréquence dépend principalement de la longueur de la cavité buccale. Ceci implique que leur valeur varie aussi en fonction du sexe de l'individu et de son type de voix dans le cas de chanteurs : basse, ténor, contre-ténor, alto, soprano, etc. Enfin, elles contribuent aussi fortement à la notion de « beauté » dans la perception de la voix. Un chanteur entrainé est capable de mettre en valeur certaines composantes du spectre de la source sonore à des fins esthétiques.[31]

Pour finir, il est important de préciser que la résonnance de la cavité buccale peut-être traduite mathématiquement sous la forme d'une fonction de transfert telle que celle présentée dans la figure 12 (résonance du conduit vocal).

31 *Ibid.*, p. 146-147.

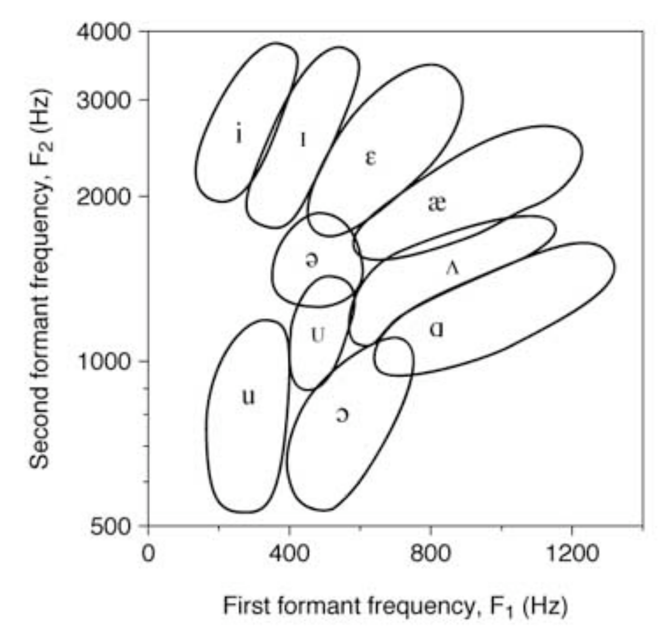


Fig. 14 – Différentes voyelles de la langue anglaise en fonction des deux premiers formants.[32]

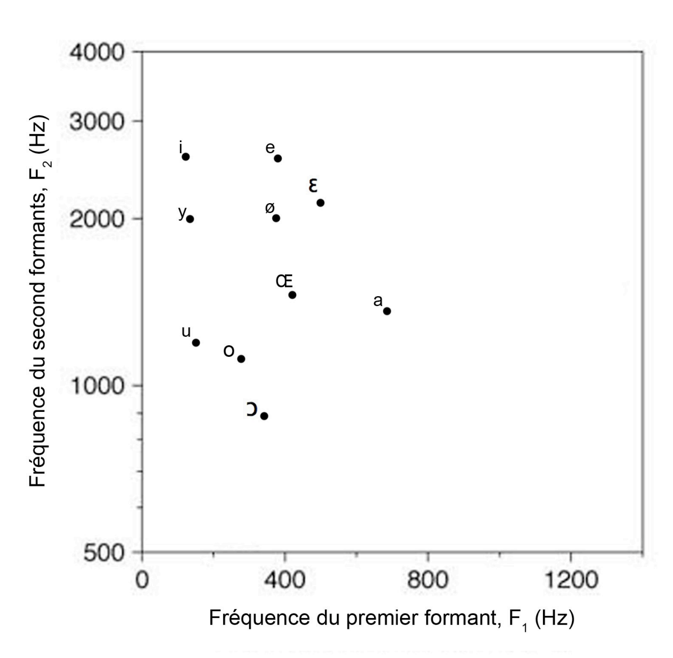


Fig. 15 – Positions des voyelles de la langue française en fonction des deux premiers formants – Adaptation des analyses de René Boite.[33]

○ La notion de diphone

Comme cela a été présenté dans le paragraphe (I)-(A)-(b), les langages sonores chez l'Homme sont composés d'un ensemble de sons. Ces sons sont combinés et enchaînés de façons différentes en fonction de ce que l'individu qui les prononce veut faire comprendre.

---

32 http://www.ncvs.org/ncvs/tutorials/voiceprod/tutorial/filter.html Site internet du National Center for Voice and Speech (NCVS), page de tutoriaux sur la productions de la voix (en ligne le 04/09/2010).

33 BOITE, René, *op. cit.*, p.16.

La notion d'enchaînement de sons dont il est ici question permet d'introduire le concept de diphone. Un diphone est un enchaînement de deux sons stables (phonèmes) avec une période d'instabilité entre les deux. Le nombre de diphones Q dans un langage est proportionnel à son nombre de phonèmes P selon la règle suivante : $Q = P^2$. La valeur de Q donnée par la formule précédente reste toutefois théorique dans la mesure où tous les phonèmes d'un langage donné ne sont pas autorisés à s'enchaîner les uns avec les autres, certaines règles devant êtres respectées.

Le programme de synthèse Diphone Studio[34] développé par l'IRCAM permet de représenter graphiquement les successions de diphones dans une phrase de la manière suivante :

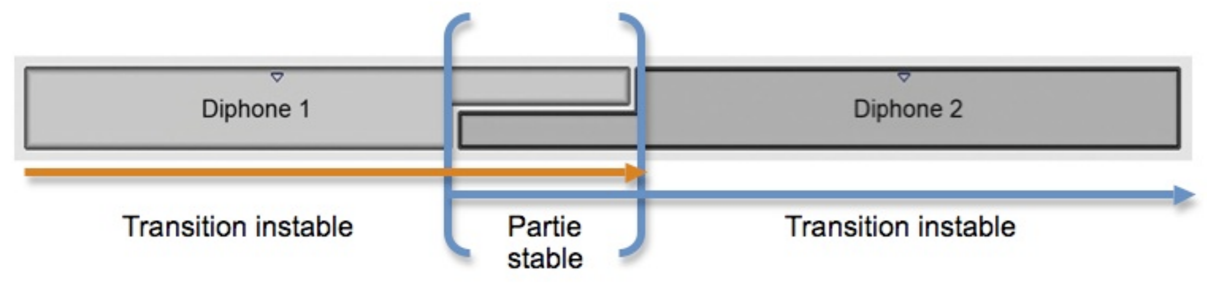


Fig. 16 – Enchaînement de deux sons stables avec diphone dans le programme Diphone Studio.

Les différentes interactions entre les organes constituants le système phonatoire autorisent la production d'un nombre très varié de sons. Ainsi l'étude de l'implication de chaque organe dans la production de la voix peut permettre la création de modèles physiques pouvant servir à sa synthèse. Dans le chapitre suivant, les différentes techniques de reproduction de la voix humaine imaginées depuis les années soixante mettant en application les informations données précédement vont être présentées.

34 http://forumnet.ircam.fr/703.html Site internet officiel du programme Diphone Studio développé par l'IRCAM (en ligne le 04/09/2010).

La synthèse numérique de la voix humaine est un domaine de recherche qui a suscité un intérêt important lors de la deuxième moitié du XXe siècle et en particulier dans les années soixante-dix et quatre-vingt. Ainsi, un grand nombre de techniques, très différentes les unes des autres ont vu le jour.

Les domaines d'application de ces techniques ont été alors très nombreux. En effet, en dehors des utilisations musicales qui sont d'ailleurs peut-être les moins connues du grand public, la voix de synthèse est un élément qui fait aujourd'hui partie intégrante de nos vies. On la retrouve aussi bien sur les boîtes vocales de nos téléphones cellulaires que dans les gares ou les transports en commun.

B. Modèles physiques basés sur la simulation de tubes acoustiques de l'appareil phonatoire : l'exemple du système SPASM (Singing Physical Articulatory Synthesis Model)

a) Fonctionnement des tubes acoustiques de l'appareil phonatoire

Un modèle physique vise à reproduire mathématiquement les différents éléments constitutifs d'un système donné. Dans le cas du système phonatoire, cette tâche se révèle être particulièrement ardue.

Le système phonatoire peut-être schématisé par un tube d'une longueur et d'une forme donnée pouvant être apparenté à d'autres instruments de musique comme ceux de la famille des cuivres. On parle alors de tube acoustique. Il est néanmoins important de préciser que sa forme est beaucoup plus complexe que ces derniers et varie au cours du temps.

La longueur d'un tube acoustique simulant l'appareil phonatoire étant significativement bien plus importante que sa largeur, il est possible de résoudre l'équation de la forme d'onde en ne considérant qu'une seule dimension.

Les tubes acoustiques de l'appareil phonatoire sont en règle générale modélisés par une succession de cylindres dont la taille est égale à la distance parcourue par l'onde sonore en une période t où t correspond à la période de la fréquence d'échantillonnage utilisée. Sur la figure 17, (a) représente la modélisation sous forme de tube acoustique faite du système phonatoire et (b) la version discrète constituée de cylindres :

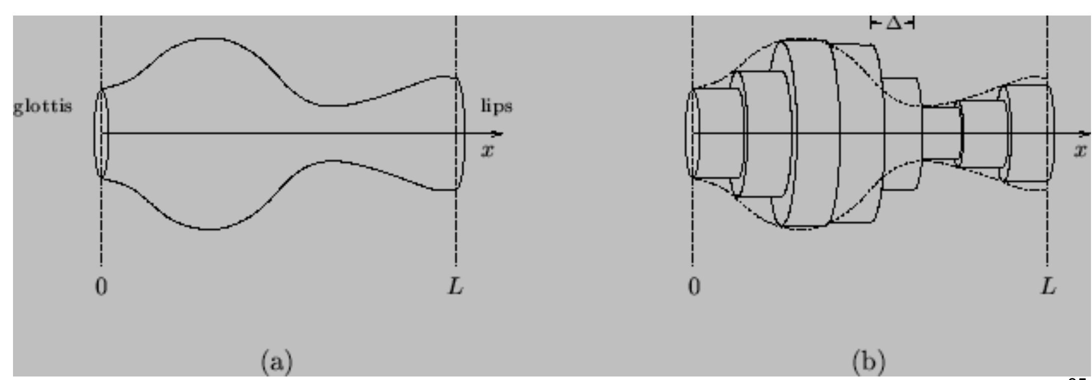


Fig. 17 – Modélisation physique sous forme de tube acoustique de l'appareil phonatoire.[35]

35 https://ccrma.stanford.edu/~bilbao/master/node5.html Page du cours « The Kelly-Lochbaum Digital Speech Synthesis Model » de Stefan Bilbao au CCRMA de Stanford (en ligne le 04/09/2010).

Cette technique fut l'une des premières utilisées pour la synthèse de la voix. En effet, la première version de tube acoustique modélisant l'appareil phonatoire fut créée en 1962 par John Kelly et Carol Lochbaum[36]. Elle ne fut jamais réellement utilisée dans la mesure où elle était trop couteuse en calcul pour les machines de l'époque.

> « Les touts premiers projets concernant la musique assistée par ordinateur aux laboratoires Bell à la fin des années cinquante donnèrent naissance à un certain nombre de systèmes pour la synthèse de la parole capable de chanter tel que le modèle par tube acoustique de Kelly et Lochbaum (1962). Il amenait les prémices des modèles physiques. A cette époque, la puissance de calcul demandée pour le faire fonctionner était telle qu'il était impossible de le commercialiser en tant que synthétiseur pour la parole. Une utilisation pour la composition musicale était aussi inenvisageable à cause des coûts de fonctionnement.[37] »

Ce modèle fut amélioré par la suite par Perry Cook qui créa le système SPASM en 1992[38].

b) Le système SPASM (Singing Physical Articulatory Synthesis Model) de Perry Cook[39]

Le système SPASM est principalement basé sur la technologie WGF (WaveGuide Filter) des guides d'ondes développée par Julius Smith[40]. Elle permet de modéliser efficacement la partie articulatoire du système vocal qui est alors contrôlée directement par des paramètres décrivant sa forme comme cela est montré dans la figure 18.

---

36 KELLY, John ; LOCHBAUM, Carol, *Speech synthesis : actes du Fourth International Congress on Acoustics, Copenhagen, septembre 1962*.

37 COOK, Perry-Raymond, « SPASM, a Real-Time Vocal Tract Physical Model Controller ; and Singer, the Companion Software Synthesis System », *Computer Music Journal*, XVII (1993), n° 1, p. 38, citation traduite de l'anglais : « The earliest computer music project et Bell Labs in the late 1950s yielded a number of speech synthesis systems capable of singing, one being the acoustic tube model of Kelly and Lochbaum (1962). This model was actually considered too computationally expensive for comercialization as a speech synthesiser, and too expensive to be practical for musical composition. ».

38 *Ibid*., p. 30-44.

39 *Ibid*., p. 30-44.

40 SMITH, Julius, *Musical Application of Digital Waveguides*, Stanford University Center for Computer Research in Music an Acoustics : Stanford Publication STAN-M-39, 1987.

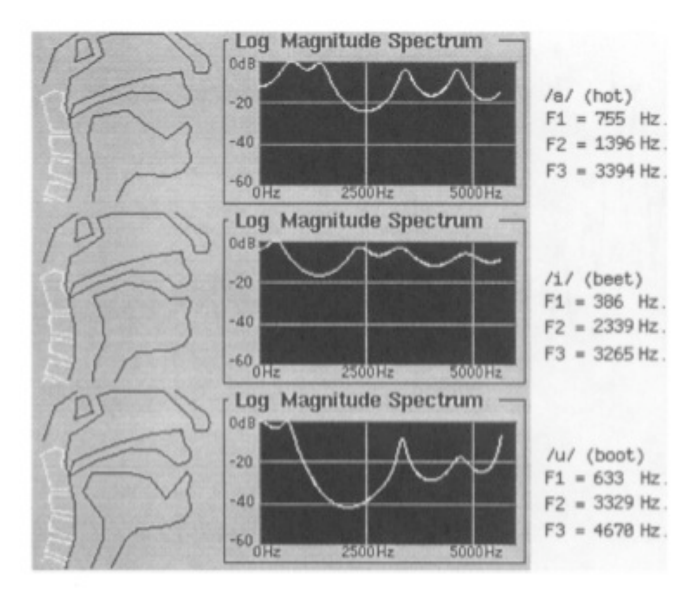


Fig. 18 – Représentation du système articulatoire réalisée avec le système SPASM pour trois voyelles différentes.[41]

Comme on peut le voir ci-dessus, SPASM inclut également la cavité nasale à son modèle du système articulatoire ce qui n'était pas le cas pour le tube acoustique de John Kelly et de Carol Lochbaum. Celle-ci peut donc être reliée au tube acoustique ou non en fonction de la position de la luette (relevée ou abaissée) exactement comme cela a été montré dans le paragraphe (I)-(A)-(c). Pour les trois voyelles présentées dans la figure 18, la luette est en position relevée, ce qui implique que la cavité nasale n'est pas prise en compte dans le calcul du son.

La source vocale qui est donc passée dans le tube acoustique est créée à partir d'un ensemble de tables de fonctions possédant des formes d'ondes différentes en fonction du type de voix désiré. La fréquence de ces tables de fonctions peut être modulée afin de créer un vibrato.

Il est aussi important de préciser que du bruit filtré peut être ajouté au son périodique de la source vocale afin de reproduire les turbulences générées lors du passage du flux d'air à travers les cordes vocales.

Enfin, une autre source de bruit filtré peut-être utilisée pour reproduire les sons de type fricatifs créés lors de la prononciation de consonnes par exemple.

Ce modèle de synthèse est particulièrement intéressant puisqu'il permet de traiter le problème « à la source ». Plus une modélisation physique est précise, plus elle

41 COOK, Perry, *op. cit.*, p. 33.

permet d’obtenir des résultats proches de la réalité. Le principal inconvénient de ce type de technique est sa complexité et donc le grand nombre de paramètres utilisés.

C. Techniques de synthèse « spectrales »

Les techniques décrites précédemment visent à reproduire un son naturel en étudiant les propriétés physiques du système qui le produit dans le but de les modéliser par des formules mathématiques.

Un autre type de technique de synthèse appelée « synthèse spectrale » passe par l'analyse du spectre du son naturel à synthétiser. Dans ce cas, il n'est donc plus question de reproduire la source d'un son, mais le son lui même avec toutes ses caractéristiques.

Dans ce chapitre, les différentes techniques de synthèse spectrale utilisées pour la synthèse de la voix vont être présentées.

a) La modulation de fréquence et l'importance du vibrato et des micros fluctuations de la fondamentale[42] [43]

La synthèse par modulation de fréquence est aujourd'hui véritablement devenue légendaire. Elle fut inventée en 1973 par John Chowning et a totalement révolutionné et démocratisé l'utilisation de la synthèse sonore dans la musique.

Sa popularité est certainement due au grand nombre d'avantages qu'elle offre à son utilisateur. En effet, sa simplicité, son nombre restreint de paramètres, la richesse des sons qu'elle permet de produire ainsi que la dynamique de leur spectre à l'instar des sons naturels furent et sont toujours ses principaux atouts.

o Technique de synthèse par modulation de fréquence

Un synthétiseur FM[44] nécessite au moins deux oscillateurs. La fréquence de l'un, que l'on appelle « le porteur », est modulée par l'autre : « le modulateur ». La fréquence du modulateur est appelée « fréquence de modulation » et son amplitude, « index de modulation ».

---

42 CHOWNING, John, « Frequency Modulation Synthesis of the Singing Voice », *Current Directions in Computer Music Research*, éd. sous la direction de Max Mathews et John Pierce, Cambridge : The MIT press, p. 57-64.

43 CHOWNING, John, *Interview Enregistrée*, Centre Interdisciplinaire d'Etude et de Recherche sur l'Expression Contemporaine (CIEREC), St Etienne, 09/11/2009, enregistrement disponible sur le CD à cd/audio/interview-chowning.m4a.

44 Frequency Modulation.

Si la fréquence du modulateur est inférieure à 20Hz, les modulations sont perceptibles dans le son créant ainsi un effet de vibrato. Au dessus de 20Hz, notre cerveau n'est plus capable de différencier chaque variation, on observe alors la formation de partiels harmoniques ou non autour de la fréquence du porteur (figure 20).

La figure 19, présente le circuit et la formule pour l'implémentation d'un synthétiseur par modulation de fréquence de base.

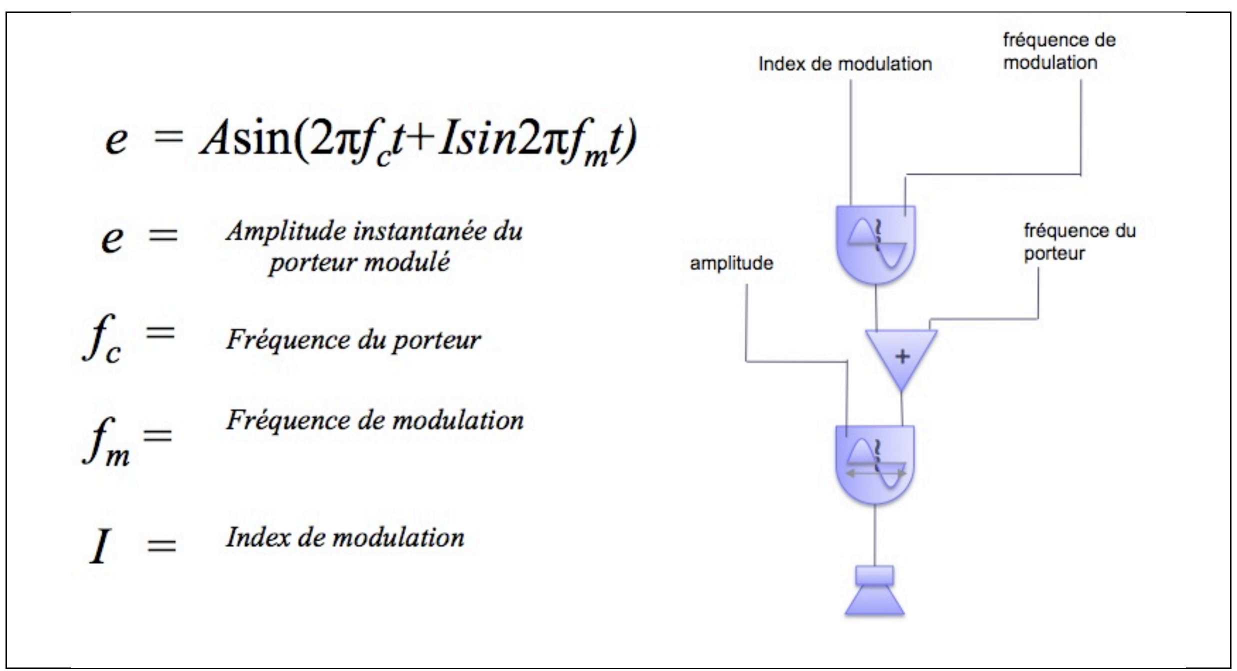


Fig. 19 – Synthétiseur par modulation de fréquence – D'après un document PowerPoint obtenu lors de l'interview de John Chowning[45]

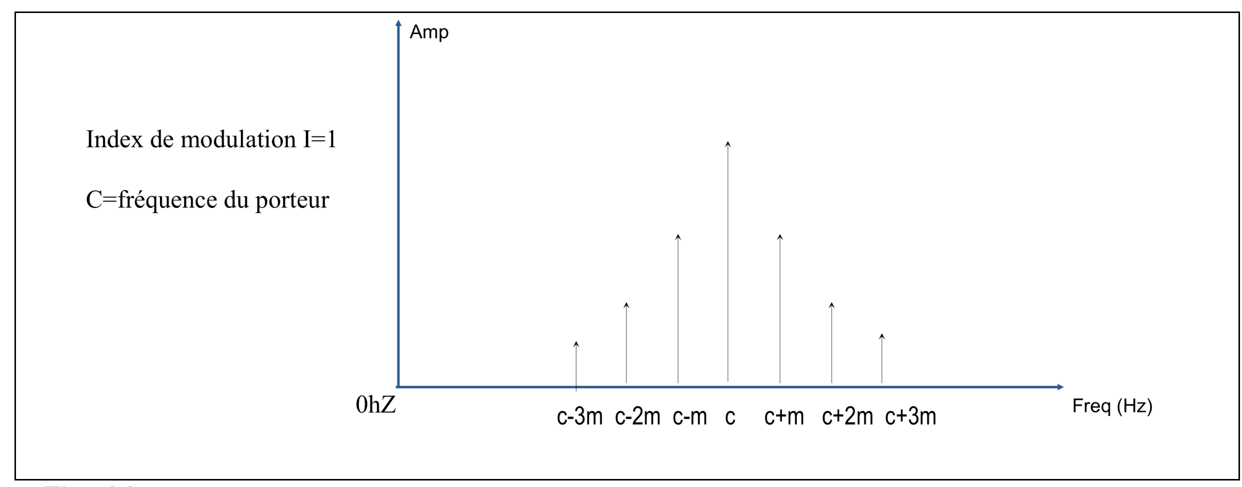


Fig. 20 – Spectre typique créé par un synthétiseur FM avec un index de modulation de 1 – D'après un document PowerPoint obtenu lors de l'interview de John Chowning[46]

45 CHOWNING, John, *Interview Enregistrée*, *op. cit*.

46 CHOWNING, John, *Interview Enregistrée*, *op. cit*.

Synthèse par modulation de fréquence appliquée à la synthèse de la voix

En 1979, John Chowning mit au point une version particulière de son synthétiseur capable de reproduire des sons de type voisés tels que des voyelles. Le modèle de synthèse reste particulièrement simple et offre des résultats surprenants. Un nombre de porteur égal au nombre de formants désirés est utilisé. Leurs fréquences et leurs amplitudes doivent avoisiner celles des formants. Ces porteurs sont modulés par le même oscillateur dont la fréquence est égale à celle de la fondamentale f_0 tel qu'il l'est montré dans la figure 21.

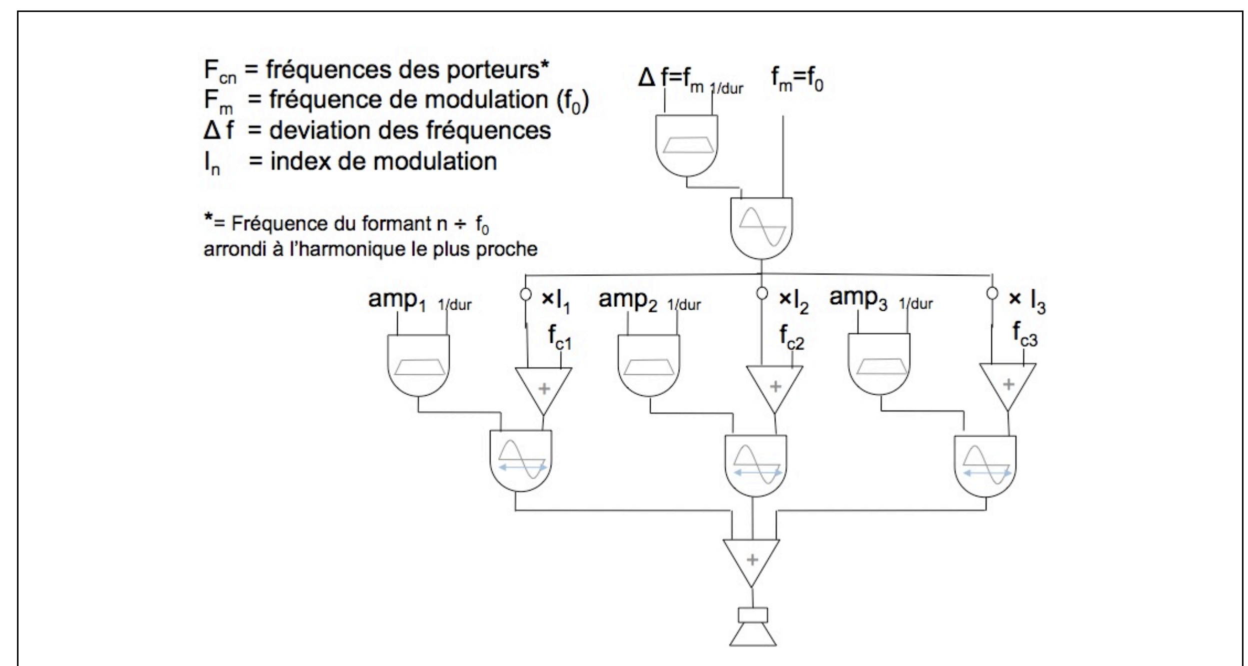


Fig. 21 – Synthétiseur vocal par modulation de fréquence – D'après un document PowerPoint obtenu de l'interview de John Chowning[47]

Importance du vibrato

Une autre grande découverte de John Chowning dans le domaine de la synthèse de la voix concerne l'importance du vibrato de la fréquence de la fondamentale. En effet, un système comme celui présenté dans la figure 21 produit un son qui n'est pas forcément rattachable à celui de la voix. Ceci est dû au phénomène de « source segregation », selon les propres termes de John Chowning.

Prenons l'exemple d'un son composé de huit partiels dont les fréquences ne changent pas au cours du temps, du même type que ceux productibles par le synthétiseur de la

47 CHOWNING, John, *Interview enregistrée*, *op. cit.*

figure 21. Un tel son paraît très « électrique » à l'oreille et peut difficilement être apparenté à celui d'une voix. En revanche, si la fréquence de chacun des harmoniques est légèrement modulée par un même oscillateur dont la fréquence est très basse (8Hz environ en règle générale), on observe alors un phénomène de fusion des harmoniques qui rend le son semblable à celui d'une voix humaine.

L'exemple sonore contenu dans le fichier FM.aif[48] permet de comprendre ce phénomène. Lors des cinq premières secondes, il est composé d'un seul harmonique : la fondamentale. Par la suite, huit autres harmoniques sont ajoutés, il est alors impossible d'apparenter ce son à celui d'une voix humaine. Enfin, à partir de la onzième seconde, les harmoniques sont modulés à une fréquence de huit Hertz avec un ratio de six pour cent, on entend une voix de femme.

- **Importance des micros fluctuations de la fondamentale f_0**

Les sons périodiques produits par les cordes vocales ne sont pas totalement parfaits et n'ont donc pas une fréquence fixe lors du maintien d'une note. En effet, l'étude du spectre des sons vocaux permet de montrer que la fréquence de la fondamentale et des différents harmoniques d'une note chantée et maintenue varie de façon infime et aléatoire au cours du temps[49].

- **Modèle avancé pour la synthèse de la voix par modulation de fréquence**

Lors de la conception de son système de synthèse vocale par modulation de fréquence, John Chowning a pris en compte ces différents facteurs et a donc ajouté un système de variation de la fondamentale à la fois périodique (vibrato) et aléatoire (microvariations de la voix) tel qu'il l'est montré dans la figure 22.
Ce modèle de synthèse fut utilisé par John Chowning dans *Phoné*[50] [51] qui souligne l'importance du vibrato dans la perception de sons vocaux.

---

48 Disponible sur le cd à cd/audio.

49 SUNDBERG, Johan, *The Science of the Singing Voice*, Dekalb (Illinois) : Northern Illinois University Press, 1987.

50 GAYOU, Evelyne, *John Chowning : portraits polychromes*, Paris : INA, 2005.

51 http://mediatheque.ircam.fr/sites/voix/creer/phone.html Présentation et analyse de Phoné de John Chowning (en ligne le 04/09/2010).

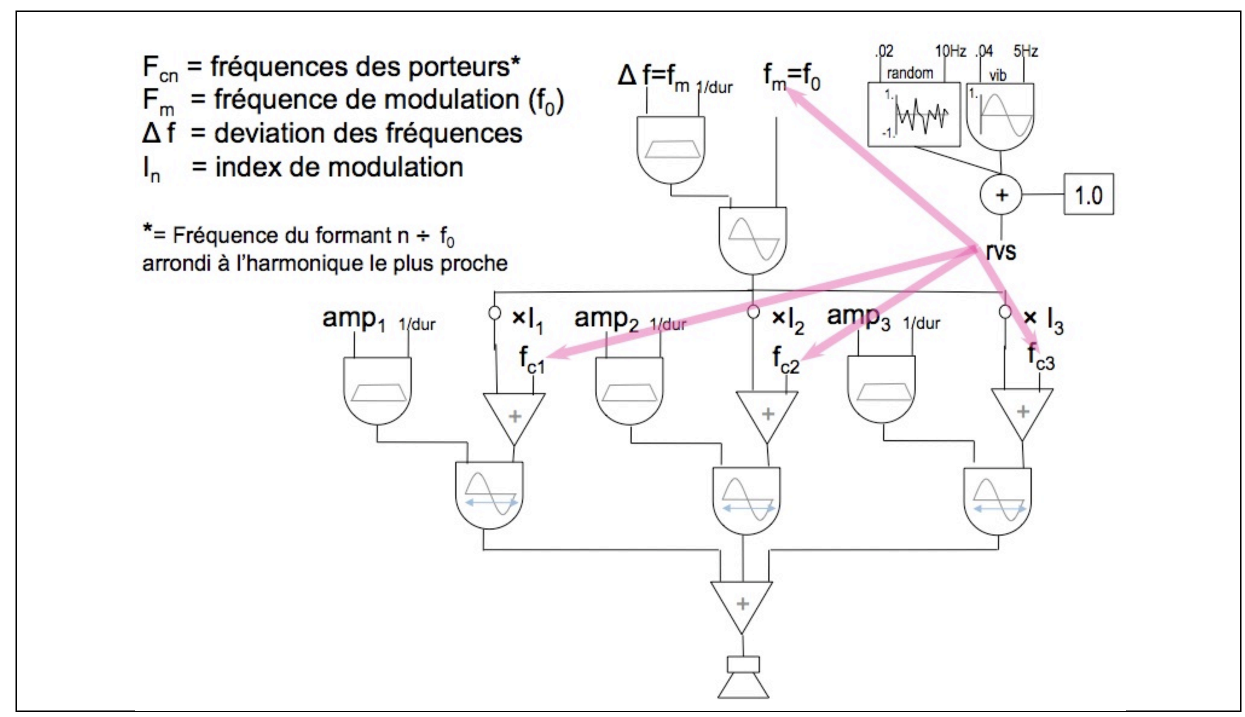


Fig. 22 – Synthétiseur vocal par modulation de fréquence avec modulation périodique et apériodique de la fréquence du fondamental et des différents formants – D'après le PowerPoint de John Chowning obtenu lors de son interview[52]

b) Analyse FFT et re-synthèse : le modèle additif[53] [54]

Comme cela a été montré dans (I)-(A)-(b), les sons produits par l'appareil phonatoire, tout comme ceux de n'importe quel autre instrument de musique sont composés de partiels, c'est à dire qu'ils peuvent être décomposés en une somme de signaux sinusoïdaux de fréquences, d'amplitudes et de phases différentes. Ils nous permettent sur le plan perceptuel de distinguer le timbre de chaque instrument et par conséquent d'identifier les sons qui nous entourent.

Le modèle additif imaginé par Robert McAulay et Thomas Quatieri en 1986[55], consiste à modéliser un son par une collection de trajets sinusoïdaux. Un trajet décrit les

---

52 CHOWNING, John, *Interview Enregistrée*, *op. cit*.

53 COOK, Perry-Raymond, « Singing Voice Synthesis : History, Current Work, and Future Directions », *op. cit.*, p. 40.

54 MAC AULAY, Robert ; QUATIERI, Thomas, « Speech Analysis/Synthesis Based on a Sinusoidal Representation », *Transaction on Acoustics, Speech and Signal Processing*, XXXIV (1986), n°4, p. 744-754.

55 Ibid.

variations dans le temps de l'amplitude, de la phase et de la fréquence d'une composante sinusoïdale du signal audio analysé.[56]

Dans un premier temps, on procède à une analyse FFT[57] du signal à re-synthétiser. Cette étape consiste à effectuer la transformée de Fourier du signal, fenêtrée par une fenêtre glissant dans le temps. L'analyse est donc répétée sur le signal temporel toutes les *t* périodes.

Par la suite, le trajet des différents partiels est détecté. Pour ceci, il est d'abord nécessaire d'extraire pour chacune des *t* analyses un même nombre maximal de partiels. Deux techniques peuvent alors être utilisées :

- L'analyse harmonique : la fréquence de la fondamentale f_0 est d'abord estimée, puis, une sinusoïde pour chaque multiple de f_0 est extraite.
- L'analyse non harmonique : les pics sinusoïdaux de plus fortes amplitudes sont sélectionnés.

Les paramètres supposés correspondre à une même sinusoïde sont ensuite connectés. Idéalement, seulement les pics sinusoïdaux sont sélectionnés (il peut y avoir confusion entre des pics de sinusoïdes et des zones bruitées) et il est fait abstraction des sinusoïdes qui ne correspondent pas à un trajet. Il est également nécessaire de prendre en compte l'apparition et la disparition éventuelle de partiels au cours du temps.

Avant de re-synthétiser le son, il est possible de modifier les paramètres issus des différentes étapes d'analyse précédentes dans le but de transposer, d'accélérer ou de ralentir le son d'origine.

La dernière étape consiste enfin à re-synthétiser le son en utilisant la technique de la synthèse additive.

---

56 CHAMPION, Gaël, *Application du modèle additif « shape invariant » pour la transformation de la voix*, Mémoire de stage de DEA ATIAM, Université Paris VI, 2004.

57 Fast Fourier Tranform.

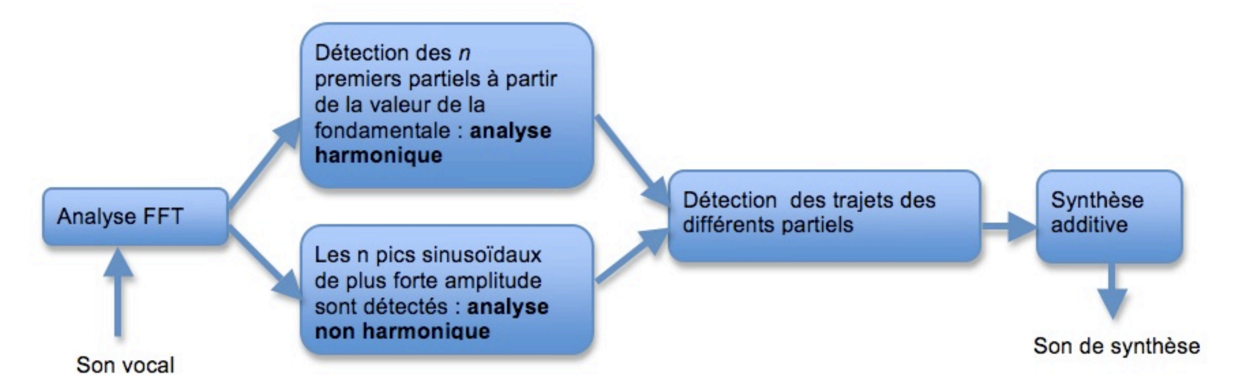


Fig. 23 – Analyse FFT et synthèse additive.

Le modèle de Robet MacAulay et de Thomas Quatieri ici présenté reste assez basique et n'était pas forcément orienté vers une utilisation pour la synthèse de la voix. En effet, il ne fonctionnait que pour des sons de type harmonique rendant alors impossible la re-synthèse des sons apériodiques des consonnes.

Il a été amélioré plus tard par d'autres équipes de recherche. On peut notament citer les travaux de Xavier Serra et Julius Smith[58] qui y ont ajouté un module de décomposition stochastique (technique appelée SMS[59]). Dans ce cas, une phase supplémentaire est ajoutée à l'étape d'analyse qui sépare les sons périodiques, qui sont synthétisés avec la technique présentée ci-dessus, des sons apériodiques (on parle alors de résidu). Ces derniers sont re-synthétisés grâce à une source de bruit filtrée.

c) Analyse et synthèse par prédiction linéaire (Linear Predictive Coding)[60] [61]

Une autre technique fonctionnant sur le modèle analyse/synthèse est la synthèse par prédiction linéaire aussi appelée LPC (Linear Predictive Coding). C'est l'une des techniques les plus utilisées pour la synthèse de la parole. Celle-ci part du principe qu'un signal vocal est produit par une source connectée à un tube dont la forme varie au cours du temps (filtre) auquel s'ajoute d'occasionnels sons apériodiques (consonnes).

Elle passe dans un premier temps par une étape d'analyse du son à synthétiser. Celle-ci consiste à prédire la valeur d'un échantillon donné en se basant sur celles des

---

58 SERRA, Xavier ; SMITH, Julius, « Spectral Modeling Synthesis: a Sound Analysis/Synthesis System Based on a Deterministic plus Stochastic Decomposition », *Computer Music Journal*, XIV (1990), n° 4, p. 12-24.

59 Sound Modeling Synthesis.

60 COOK, Perry-Raymond, « Singing Voice Synthesis : History, Current Work, and Future Directions », *op. cit.*, p. 39.

61 ATAL, Bishnu, « Speech Analysis and Synthesis by Linear Prediction of the Speech Wave », Journal of the Acoustical Society of America, XLVII (1971), n°65(A), p. 637-645.

échantillons précédents. Dans le cas d'un son vocal, le but est d'estimer la valeur de chaque formant à un instant donné et de supprimer leurs effets sur le son analysé. On parle alors de filtrage inverse (inverse filtering). Le son produit lors de cette opération est appelé résidu. La dernière phase de cette étape d'analyse consiste à estimer l'intensité et la fréquence du résidu.

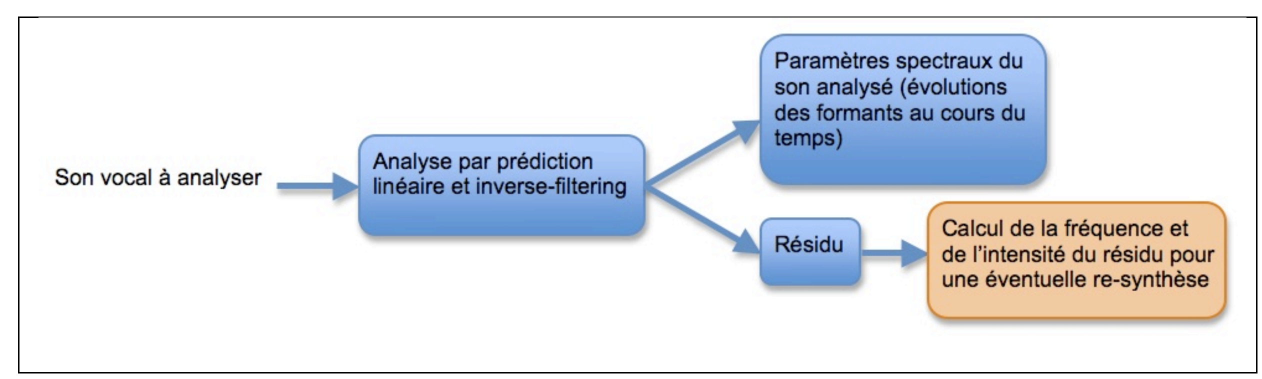


Fig. 24 – Analyse par prédiction linéaire d'un son vocal – Les étapes obligatoires sont représentées par les rectangles bleus, les étapes facultatives par les rectangles oranges.

La synthèse effectuée à partir de cette analyse est alors très simple à mettre en place. En effet, la réponse du filtre agissant sur la source vocale étant connue, il suffit de l'appliquer sur le signal créé par un générateur d'impulsions produisant un son semblable à celui des cordes vocales dont les paramètres sont la fréquence et l'intensité calculées à partir du résidu (dans le cas où l'on souhaite re-synthétiser le son d'origine). Pour les sons vocaux de type fricatif, le générateur d'impulsion est remplacé par un générateur de bruit.

Le son émis lors de la parole étant dynamique et changeant de façon constante, il est nécessaire de reproduire les opérations précédentes n fois par seconde (généralement, $2 \leq n \leq 6$).

Ce modèle de synthèse peut être résumé de la façon suivante :

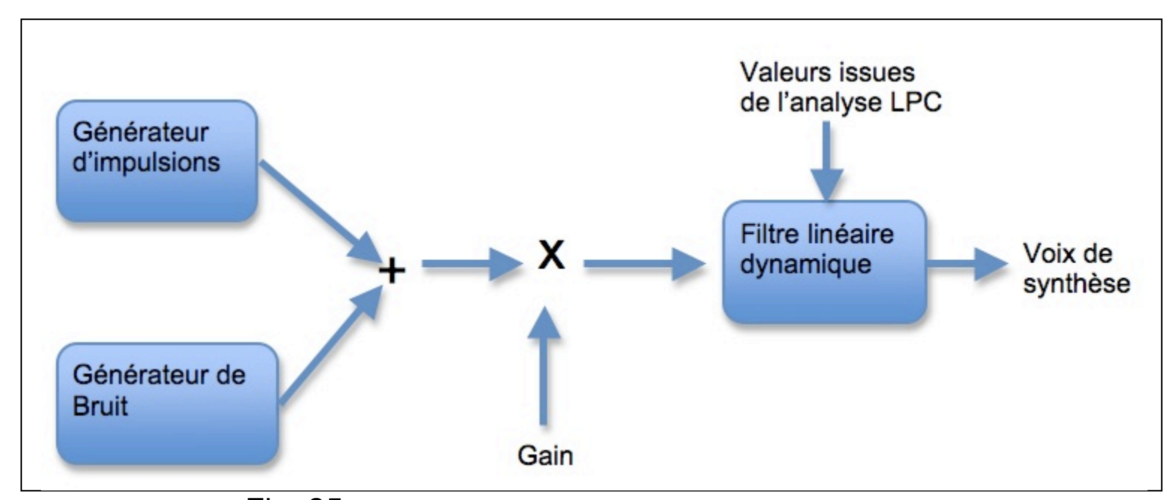


Fig. 25 – Synthèse d'un son vocal par prédiction linéaire.

L'examen de la figure précédente permet immédiatement de faire une parenté indirecte entre la technique de synthèse par prédiction linéaire et un modèle physique de l'appareil phonatoire tel que celui présenté dans (I)-(B)-(a).

Le fait que cette technique parte du principe que le système phonatoire est linéaire est un de ses principaux inconvénients. En effet, certains effets aérodynamiques ainsi que certaines mises en résonance des cordes vocales peuvent amener à l'apparition de comportements non linéaires du son qui ne peuvent donc pas être perçus par l'analyse.

La LPC a été abondamment utilisée dans la musique techno dans les années quatre-vingt-dix où des sons d'instruments de musique étaient utilisés pour remplacer le générateur d'impulsions du système décrit précédemment. Ainsi, les propriétés spectrales du son vocal analysé (formants, etc.) étaient appliquées au sons d'instruments utilisés.

Le champ d'application de la LPC dépasse celui de la synthèse sonore. En effet, cette technique est également fortement utilisée dans le domaine des télécommunications pour la compression des signaux vocaux. Elle est par exemple utilisée dans le standard GSM[62].

---

62 Global System for Mobile.

D. Techniques de synthèse basées sur la concaténation d'échantillons

Une autre manière de reproduire de façon artificielle des sons naturels est l'utilisation de techniques d'échantillonnage.

L'augmentation de la taille de la mémoire des ordinateurs au cours des trente dernières années ainsi que la demande de plus en plus forte des musiciens pour avoir des outils numériques permettant l'imitation d'instruments de musique sont des facteurs qui ont fortement contribués au développement des méthodes d'échantillonnage et à leur commercialisation. Ces dernières permettent de reproduire facilement le son d'instrument de musique au détriment toutefois du contrôle sur le résultat. Elles se différencient donc sur ce dernier point des techniques présentées précédemment qui permettent un contrôle accrus sur le résultat obtenu.

Les techniques d'échantillonnage utilisées pour la synthèse de la voix sont principalement basées sur la modification (transposition, filtrage, etc.) et la concaténation d'échantillons. Leur fonctionnement va être décrit dans ce chapitre.

a) Modèles par « performance sampling » : sélection automatique, concaténation et modification d'échantillons[63]

L'imitation par synthèse d'un instrument de musique par une méthode d'échantillonnage passe d'abord par la création d'une base de données d'échantillons sonores. Dans le but d'obtenir de bons résultats, ces échantillons doivent être représentatifs de la phonétique complète de l'instrument de musique, c'est à dire correspondre à l'ensemble des sons productibles par cet instrument lorsqu'il est manipulé par un musicien. Le domaine considéré est alors parfois très grand voire même infini. Ce n'est pas le cas de l'échantillonnage où seul un nombre fini d'échantillons peut-être utilisé. Pour cette raison, les échantillons sélectionnés doivent être significatifs des états particuliers de l'instrument. Ainsi, les variables peuvent être très diverses : fréquence, timbre, amplitude, mode de jeux, etc.

---

63 BONADA, Jordi ; SERRA, Xavier, « Synthesis of the Singing Voice by Performance Sampling and Spectral Models », *Signal Processing Magazine*, XXIV (2007), n°2, p. 67-79.

Un échantillon peut-être représenté sous la forme d'une trajectoire dans l'espace sonore d'un instrument de musique :

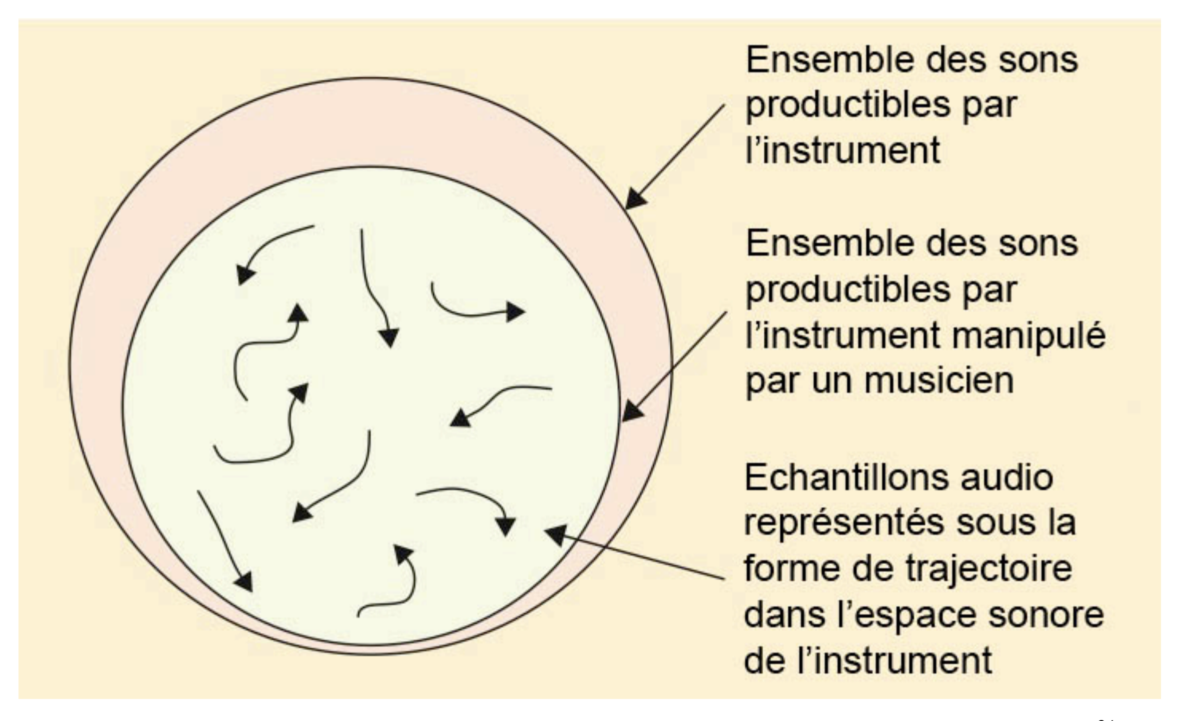


Fig. 26 – Représentation et échantillonnage de l'espace sonore d'un instrument de musique[64] – Le premier ensemble (en rose) correspond à l'intégralité des sons productibles par l'instrument. En jaune, on peut voir le sous-ensemble correspondant aux sons productibles par l'instrument lorsqu'il est manipulé par un musicien. La taille de ce dernier varie en fonction de chaque musicien et de leur niveau. Les dix flèches représentent des échantillons audio enregistrés de l'instrument sous forme de trajectoire dans l'espace sonore.

Afin de modéliser l'intégralité des sons contenus dans le sous-ensemble jaune de la figure 26 avec un nombre fini d'échantillons, il est nécessaire de les modifier en fonction du résultat souhaité comme le montre la figure suivante :

---

64 *Ibid.*, p. 68.

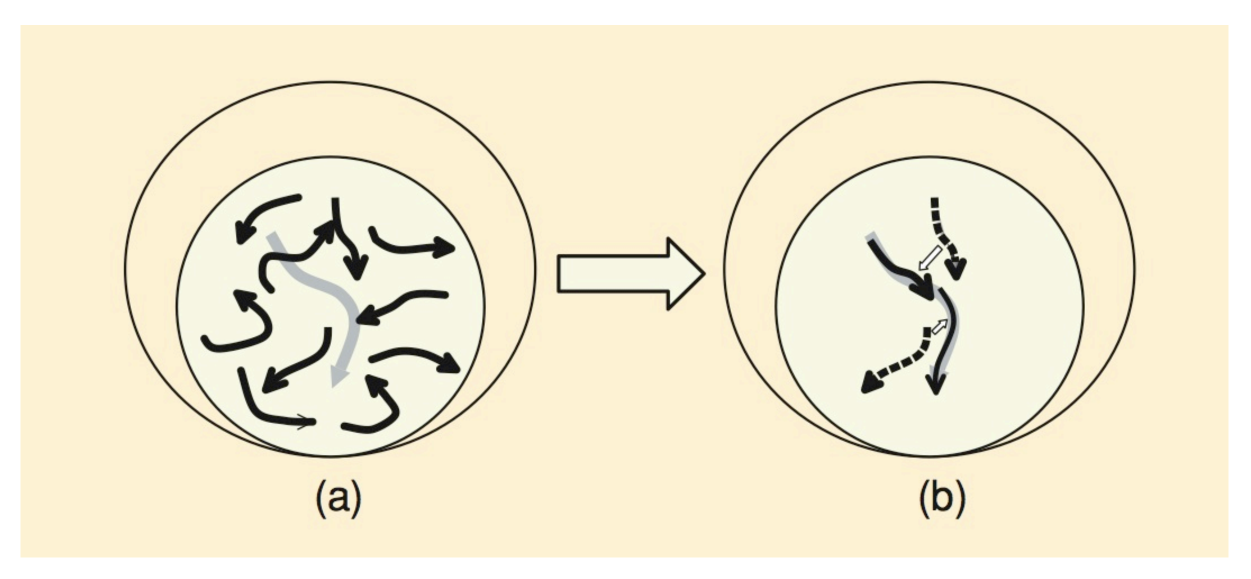


Fig. 27 – Modification de deux échantillons afin d'obtenir un résultat inédit[65] – Dans (a), l'espace sonore de l'instrument de musique est modélisé par un ensemble de 10 échantillons (flèches noires). Le son désiré est représenté sous forme d'une trajectoire dans cet espace sonore (flèche grise). Dans (b), le synthétiseur modifie les deux échantillons les plus proches de la trajectoire désirée afin de s'en rapprocher le plus possible.

La représentation donnée précédemment reste très schématique. Ainsi, l'équipe de Xavier Serra et de Jordi Bonada[66] a mis au point une méthode de synthèse par échantillonnage adaptée à la voix. La base de données contenant les échantillons divise l'espace sonore de la voix en trois sous-ensembles[67] :

- Le premier considère un échantillon donné sous quatre aspects : son tempo, son intensité, sa hauteur et le phonème utilisé.
- Le second permet de différencier les échantillons en fonction du type de vibrato utilisé : léger, sec, épais, lyrique, etc.
- Le troisième sous-ensemble considère un échantillon donné en fonction de son type d'articulations musicales : legato, portamento, attaque piquée, etc.

Les informations nécessaires pour la synthèse : type de voix, syllabes, nuances, vibrato etc. sont contenues dans une partition qui est fournie au synthétiseur. Elle est alors traitée par un module qui la transforme en listes de paramètres de plus bas niveau utilisables pour la synthèse (liste de hauteurs, liste d'articulations, liste d'intensités, type de vibrato, etc.)[68]. Les échantillons les plus proches du résultat à obtenir sont sélectionnés et modifiés en fonction des paramètres donnés dans l'étape précédente. Les modifications sont effectuées à

65 *Ibid*. p. 78.

66 *Ibid*., p. 67-79.

67 *Ibid*., p. 74-76.

68 *Ibid*., p. 76 - 77.

l'aide de méthodes d'analyse et de re-synthèse telles que celles présentées dans (I)-(C)-(b) et (I)-(C)-(c) permettant par exemple de transposer ou de modifier le tempo d'un son.

Afin de synthétiser une phrase musicale complète, il est nécessaire de concaténer les échantillons modifiés. Pour que le résultat sonne le plus naturel possible et pour éviter les discontinuités, il est nécessaire d'interpoler les différentes composantes spectrales de chaque échantillon. En effet, les échantillons subissent tous une re-synthèse lors de leur modification. Il est donc possible d'utiliser la technique de suivi de trajectoire des différentes composantes sinusoïdales d'un son présentée dans (I)-(C)-(b) pour interpoler chaque échantillon.

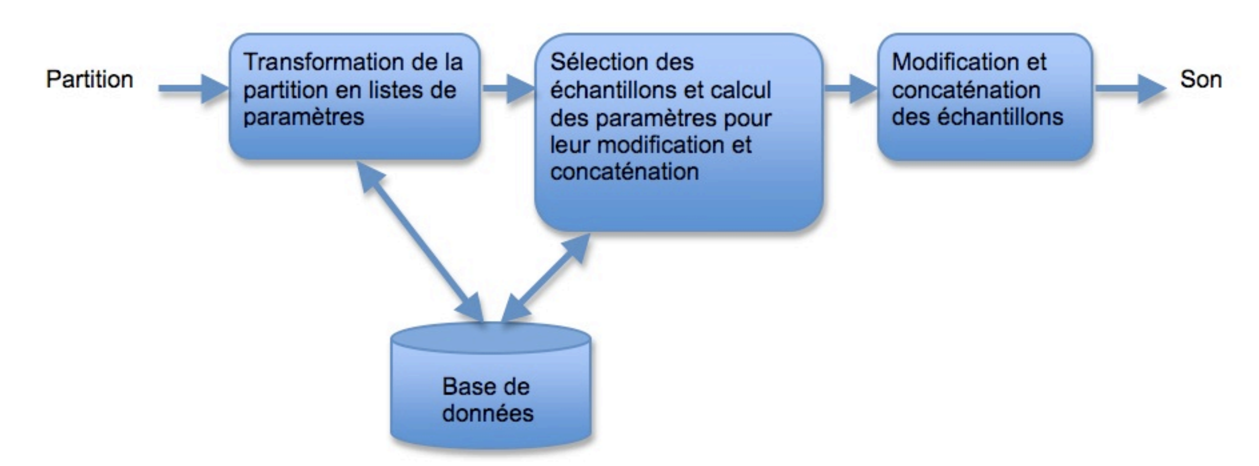


Fig. 28 – Echantillonnage par « performance sampling ».[69]

Cette technique a été commercialisée par Yamaha en 2004 sous le nom du programme Vocaloid[70]. Les résultats obtenus peuvent être entendus dans la vidéo vocaloid.mp4[71].

b) Pitch Synchronous Overlap Add (PSOLA)[72]

La plupart des méthodes de synthèse par échantillonnage passent par une étape de modification des échantillons. Par exemple, dans un cas hypothétique très simplifié où l'on veut synthétiser le son d'un piano alors qu'on ne dispose que de deux échantillons de

69 *Ibid*., p. 69.

70 http://www.vocaloid.com/ Site officiel non commercial du programme « vocaloid » développé par Yamaha (en ligne le 04/09/2010).

71 *Close your eyes*, chanté par Kaito et Meiko, les voix virtuelles créées par Yamaha, disponible sur le cd cd/video/vocaloid.mp4, vidéo téléchargée depuis le site YouTube à l'adresse suivante : http://www.youtube.com/watch?v=cVU-tWu9LoY (en ligne le 04/09/2010).

72 RODET, Xavier ; MANOURY, Philippe ; LEMOUTON, Serge ; PEETERS, Geoffroy ; SCHNELL, Norbet, *Synthesizing a choir in real-time using Pitch Synchronous Overlap Add (PSOLA) : actes du First Benelux Workshop on Model Based Processing and Coding of Audio, Bruxelles, 2002*, Paris : IRCAM, 2002.

l'instrument, il faudra forcément modifier leur hauteur si l'on veut être en mesure de reproduire la tessiture totale de l'instrument. La méthode la plus simple utilisée dans ce cas de figure consiste à accélérer ou à ralentir la vitesse de lecture de l'échantillon afin de modifier sa hauteur. Le gros inconvénient de cette méthode est qu'en plus de changer la hauteur de l'échantillon, on modifie aussi sa durée. Cet effet secondaire, qui est acceptable pour la synthèse de certains instruments est toutefois très contraignant pour la synthèse de la voix. En effet, pour la voix comme pour un certain nombre d'instruments, quand on joue différentes notes, certaines composantes fréquentielles du son sont proportionnelles à la transposition, d'autres restent fixes, comme c'est généralement le cas pour les formants de la voix par exemple. Ainsi, certains modèles d'échantillonnage utilisent la technique de Pitch Synchronous Overlap Add[73] plus communément appelée PSOLA qui permet de contrôler indépendamment la fréquence fondamentale f_0, la vitesse de lecture et la position des différents formants d'un échantillon de son vocal.

PSOLA est basé sur le modèle analyse/modification/re-synthèse tout comme celui utilisé dans les parties (I)-(C)-(b) et (I)-(C)-(c). Pour fonctionner correctement, cette technique ne peut être appliquée qu'à des sons harmoniques périodiques.

Comme cela a été montré dans la partie (I)-(A)-(b), le son produit par l'appareil phonatoire est en partie composé de sons périodiques bien qu'il contienne aussi des signaux apériodiques comme les sons de type fricatif. La première étape de l'analyse PSOLA consiste donc à isoler les sons périodiques de ceux qui ne le sont pas. Des marqueurs sont alors placés sur la partie périodique extraite du son d'origine à intervalles réguliers toutes les $T0_i$ périodes et si possible sur chaque zone où l'énergie est maximale ($t_{l,i}$ sur la figure 29). L'estimation de la période et des points d'énergie maximale peut se faire de la façon suivante :

---

73 CHARPENTIER, Francis ; MOULINES, Eric, « Pitch Synchronous Waveform Processing Techniques for Text-to-Speech Synthesis Using Diphones », *Speech Communication*, IX (1990), p. 453-467.

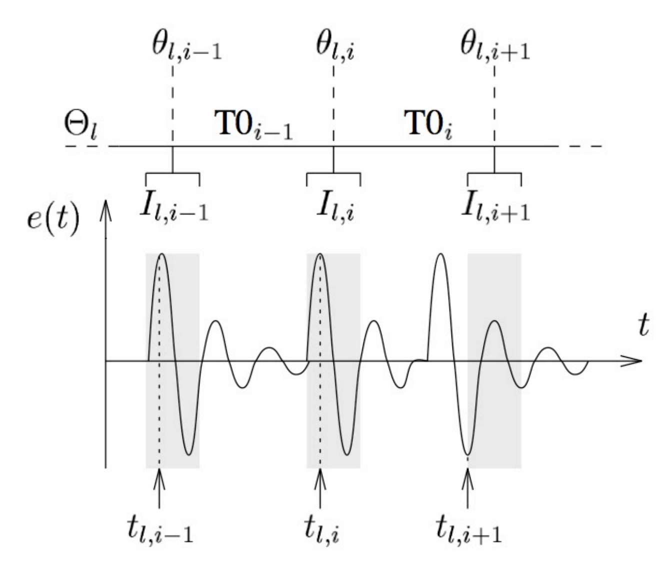


Fig. 29 – Estimation des zones d'énergie maximale à intervalle de temps régulier (en grisé) où $T0_i$ est la période et $t_{l,i}$ les points d'énergie maximale.[74]

Sur la figure 30, le signal est découpé au niveau des différents marqueurs m_i en respectant au maximum les deux conditions données précédemment (période et points d'énergie maximale). Si l'on remet bout à bout dans l'ordre chacun des morceaux découpés, il est possible de reconstituer le son d'origine sans aucune perte. Pour modifier la fréquence de la fondamentale f_0 de l'échantillon sans affecter la durée, il suffit de changer la distance entre chacun des morceaux et donc la période T0 qui devient T(t) sur la figure suivante :

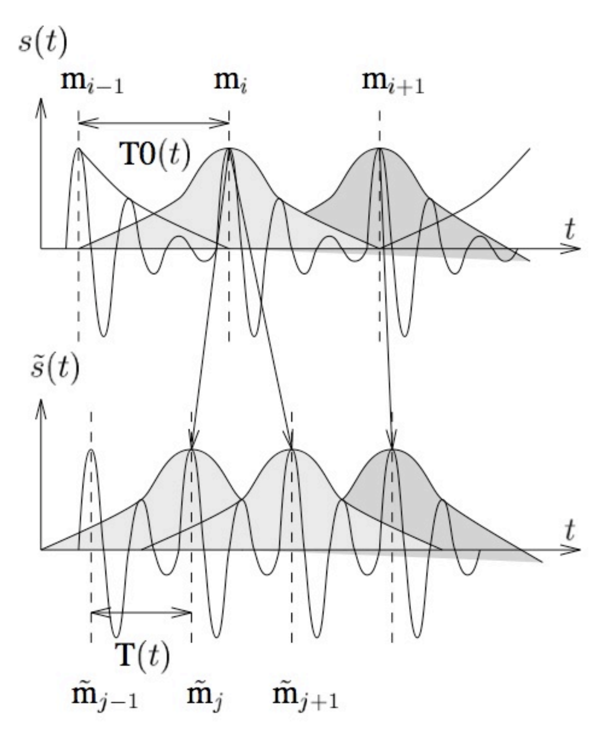


Fig. 30 – Exemple de modification de la fréquence f_0 avec la technique PSOLA – Le graphique supérieur représente le son d'origine, le graphique inférieur le son modifié.[75]

---

74 RODET, Xavier ; MANOURY, Philippe ; LEMOUTON, Serge ; PEETERS, Geoffroy ; SCHNELL, Norbet, *op. cit.*

La technique PSOLA a été améliorée par un certain nombre de travaux de recherche. On peut par exemple citer le système SINOLA mis au point par Xavier Rodet et Geoffroy Peeters à l'IRCAM faisant l'amalgame des méthodes de synthèse additive étudiées dans la partie (I)-(C)-(b) et de PSOLA[76].

Enfin, on peut noter que PSOLA a été utilisé lors de la création de l'opéra *K*... de Philippe Manoury[77] qui était alors assisté par Serge Lemouton. Cet exemple est assez particulier dans la mesure où la synthèse de la voix fut ici utilisée pour synthétiser un chœur[78].

c) L'exemple de la bande originale du film de Gérard Corbiau *Farinelli, il castrato*[79]

Bien que de plus en plus efficace, les modèles de synthèse et d'échantillonnage existant ne sont pas encore en mesure de reproduire à la perfection une voix naturelle. C'était donc encore moins le cas en 1994, date de sortie du film *Farinelli, il castrato* de Gérard Corbiau.

Ce film, met en scène la vie d'un des plus fameux castrat de l'histoire de la musique. Le principal enjeu technique du film fut la reproduction fidèle d'une voix de castrat de l'époque. En effet, la castration humaine étant interdite depuis le XIXème siècle, le dernier castrat connu mourût en 1922.

Les castrats possédaient des aptitudes vocales qu'aucun chanteur n'est aujourd'hui en mesure d'acquérir. Ils étaient capables de chanter sur plus de trois octaves et demie et pouvait maintenir des sons pendant plus d'une minute du fait de leur système respiratoire d'adulte et de leurs cordes vocales d'enfant.

La voix du castrat dans le film Farinelli est donc une voix artificielle créée grâce à l'amalgame d'une voix de contre-ténor et de soprano coloratur. La technique utilisée fut mise au point par l'équipe de Philipe Depalle[80] à l'IRCAM.

---

75 Ibid.

76 Ibid.

77 SCEMAMA, Patrick, *K..., opéra en douze scènes de Philippe Manoury*, Paris : Opéra national de Paris, 2000.

78 RODET, Xavier ; MANOURY, Philippe ; LEMOUTON, Serge ; PEETERS, Geoffroy ; SCHNELL, Norbet, *op. cit*.

79 Extrait du making-of du film *Il castrato*, vidéo disponible sur le cd dans le fichier « farinelli.mp4 » à cd/vidéo et sur le site YouTube à l'adresse suivante : http://www.youtube.com/watch?v=No-w5l00pho (en ligne le 04/09/2010).

80 DEPALLE, Philipe ; GARCIA, Guillermo ; RODET, Xavier, *A Virtual Castrato : acte de l'International Computer Music Conference, Aarhus (Danemark), octobre* 1994, http://articles.ircam.fr/textes/Depalle94a/ (en ligne le 04/09/2010).

La bande originale du film fut d'abord enregistrée avec l'orchestre et les deux chanteurs. Le contre-ténor chantait les parties graves du castrat et la soprano les parties hautes. Lors de l'interprétation, ils se relayaient donc, chantant parfois seul pour les zones extrêmes de la tessiture du castrat et ensemble pour les zones intermédiaires. Lors de cette étape, les deux chanteurs durent effectuer un important travail sur leur voix afin que leur timbre et leur interprétation soient en adéquation.

A partir des enregistrements effectués, une base de données fut créée avec la voix du contre-ténor. Cette base de données contenait toutes les combinaisons possibles de fréquences de la fondamentale, d'intensités et de voyelles caractérisant son interprétation. Par la suite, la phrase musicale à traiter devait être segmentée afin de créer des portions caractérisées par l'un des deux chanteurs, la voyelle utilisée, la fréquence de la fondamentale, l'intensité et le début et la fin de la portion. Les résultats de cette étape d'analyse permettaient alors la sélection automatique d'un des échantillons de la voix de contre-ténor dans la base de données. La phase de « morphing » pouvait alors être entamée. L'idée principale de cette technique consiste en la modification de la voix de soprano afin de lui donner les caractéristiques de la voix de ténor par l'utilisation d'un vocodeur de phase.

Pour finir, des ajustements tels que l'atténuation de certaines hautes fréquences du spectre furent exécutés sur le son résultant des opérations précédentes afin de donner une aspect plus juvénile à la voix créée.

Le modèle utilisé peut être résumé dans le block diagramme suivant :

Fig. 31 – Block diagramme du modèle d'échantillonnage utilisé pour la synthèse de la voix de Farinelli dans le film *Farinelli, il castrato* de Gérard Corbiau.[81]

81 DEPALLE, Philipe ; GARCIA, Guillermo ; RODET, Xavier, *A Virtual Castrato : acte de l'International Computer Music Conference, Aarhus (Danemark), octobre* 1994, http://articles.ircam.fr/textes/Depalle94a/ (en ligne le 04/09/2010).

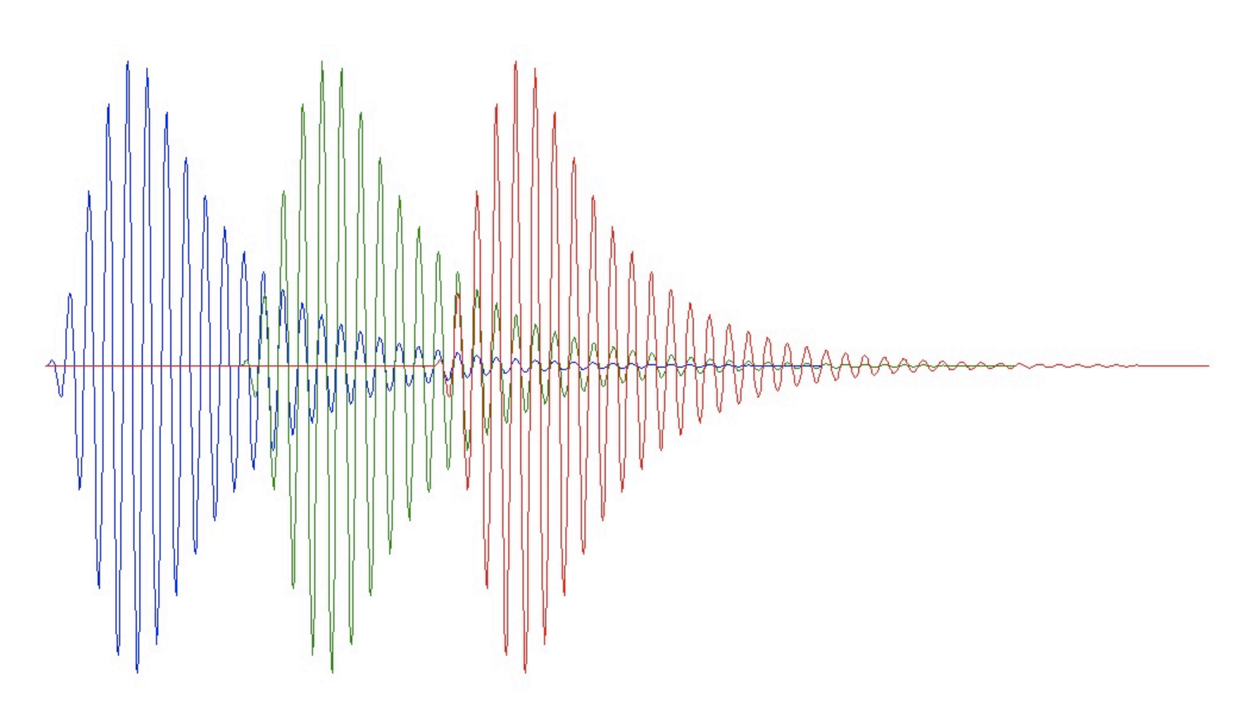
Le modèle présenté ici n'est pas réellement une technique d'échantillonnage à proprement parler comme celle de (I)-(D)-(a). En effet, il est adapté à un cas de figure bien particulier et nécessite l'interprétation des phrases musicales à synthétiser par de vrais chanteurs. Le résultat obtenu est néanmoins très impressionnant et de bien meilleure qualité que celui entendu dans la partie précédente : exemple audio contenu dans « farinelli.aiff »[82].

La synthèse de la voix chantée est un domaine qui a fait d'importants progrès au cours des quarantes dernières années principalement à cause de l'évolution rapide des technologies numériques lors de cette période. Néanmoins, la voix humaine reste un instrument de musique d'une très grande complexité. Les paramètres à prendre en compte sont tellement divers et variés qu'il est particulièrement complexe de reproduire des sons vocaux sur un ordinateur.

Bien que les résultats obtenus de nos jours soient encourageants, on est encore loins de pouvoir comparer une voix chantée de synthèse à une voix chantée naturelle. Ainsi, un grand nombre d'éléments dans les modèles de synthèse présentés précédemment nécessitent d'être améliorés. Comme cela été expliqué au cours de ce chapitre, des progrès pourraient par exemple être faits au niveau de la synthèse des sons fricatifs de la voix et de l'expressivité. Sur ce dernier point, il serait intéressant d'améliorer l'automatisation de la production des paramètres pour la synthèse à partir d'une partition. Enfin, un travail important reste à effectuer au niveau de la création d'interfaces graphique adaptées à l'analyse et à la synthèse des sons vocaux notamment au niveau de la représentation de certaines composantes du son : vibrato, etc.

82 Disponible sur le cd à cd/audio.

II. La synthèse par fonctions d'onde formantique et le programme CHANT



Succession de Fonctions d'Onde Formantique (cf. figure 32) - D'après le script Matlab 'fofd.m' disponible sur le cd à cd/matlab et dans l'annexe n°1.

Bien que la plupart des techniques utilisées pour la synthèse de la voix aient été brièvement décrites dans les chapitres précédents, il reste à présenter un dernier modèle : la synthèse par Fonctions d'Onde Formantique ou FOF. Une attention particulière lui est accordée ici puisqu'elle fait l'objet d'une implémentation complète dans le chapitre III.

Cette technique de synthèse spectrale fut fortement utilisée dans les années quatre-vingt et au début des années quatre-vingts-dix à travers le programme CHANT. Elle permet, au même titre que la synthèse par modulation de fréquence de John Chowning, d'obtenir de très bons résultats sans demander trop de calculs à la machine. Ce détail était particulièrement important à une époque où les ordinateurs étaient moins puissants que de nos jours.

Dans ce chapitre, les caractéristiques techniques détaillées d'un synthétiseur par Fonctions d'Onde Formantique vont être données. Une description complète du programme CHANT qui implémente cette technique de synthèse sera faite aussi.

A. Synthèse par Fonctions d'Onde Formantique (FOF)[83] [84]

a) Construction d'un signal à partir de FOFs

La synthèse par Fonctions d'Onde Formantiques ou FOF a été créée au début des années quatre-vingt à l'IRCAM par l'équipe dirigée par Xavier RODET.

Cette technique permet d'obtenir un résultat semblable à celui donné par un modèle traditionnel de type source/filtres mais nécessite un nombre réduit de calculs et fournit un contrôle plus intuitif et naturel des paramètres. Elle consiste à faire la somme des signaux élémentaires correspondant à la réponse impulsionnelle de filtres résonants[85] du type : $H(z) = \sum_{i=1}^{J} \frac{1 + d_i z^{-1}}{1 + a_i z^{-1} + b_i z^{-2}}$ avec a et b qui contrôlent la fréquence centrale et la largeur de bande du filtre, c son amplitude et d son inclinaison. J est le nombre de FOF équivalent aux J filtres du second ordre disposés en parallèle.

---

83 RODET, Xavier, « Time-Domain Formant-Wave-Function Synthesis », *Computer Music Journal*, VIII (1984), n° 3, p. 9-14.

84 RODET, Xavier ; BARRIERE, Jean-Baptiste ; POTARD, Yves, *Rapport de recherche n°35 : Chant, de la synthèse de la voix à la synthèse en général*, Paris : IRCAM, 1985.

85 POTTIER, Laurent, « Le contrôle de la synthèse sonore par ordinateur », *Le Calcul de la Musique*, éd. sous la direction de Laurent Pottier, Saint-Etienne : Publication de l'Université de Saint-Etienne, 2009, p. 278-279.

Ainsi, il est possible de diviser un signal *E(k)* en un nombre *J* de FOFs correspondant à *J* régions du spectre. Si le signal *E(k)* est une succession d'impulsions : $E(k) = \sum_{-\infty}^{+\infty} p_n(k)$ où *n* indexe les impulsions successives, alors la réponse *S* du filtre précédent est la somme des réponses $S_n(K)$ décalées d'une période du fondamental $T = 1/F_0$ (où F_0 est la fréquence fondamentale de l'excitation et de la réponse).

Cette même réponse $S_n(K)$ est aussi la somme des *J* réponses partielles des cellules disposées en parallèle : $s(k) = \sum_{i=1}^{J} S_{n.i}(k)$ où les $S_{n.i}(K)$ sont des Fonctions d'Onde Formantique dans la mesure où elles correspondent de manière globale aux formants du système.

Pour modifier la fréquence fondamentale du signal, il est possible de changer les durées de périodes fondamentales $T=1/F_0$.

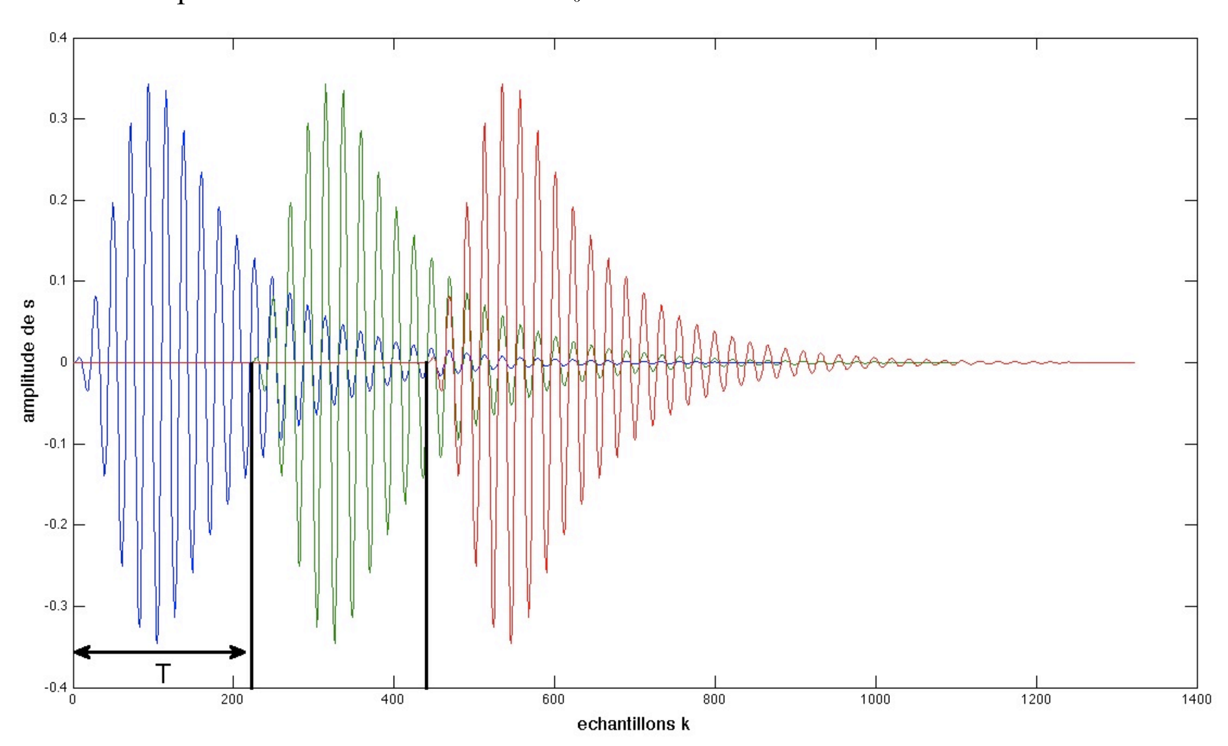


Fig. 32 – Déclenchement périodique des FOFs toutes les *T* secondes – D'après le script Matlab 'fofd.m'[86].

Chaque FOF correspondant à un formant du signal produit, il est particulièrement aisé de contrôler son spectre et ce avec un nombre réduit de paramètres.

---

86 Disponible sur le cd à cd/matlab et dans l'annexe n°1.

b) Composition des FOFs

Une FOF est le résultat d'une convolution entre une sinusoïde et une enveloppe exponentielle amortie dont la discontinuité initiale est lissée sur une durée de π/β échantillons :

$s(k) = 0$ pour : $k<0$ (1)

$s(k) = 1/2\ (1 - \cos(\beta k)) * e^{-\alpha k} \sin(wk + \phi)$ pour : $0 \leq k \leq \pi/\beta$ (2)

$s(k) = e^{-\alpha k} \sin(wk + \phi)$ pour $\pi/\beta < k$ (3)

où w est la fréquence centrale du maximum (en Hertz), α la largeur de bande à -3 db (en Hertz) et π/β la largeur des jupes. Cette dernière variable correspond en fait à la durée de l'attaque de l'impulsion qui s'exprime donc en ms (2).

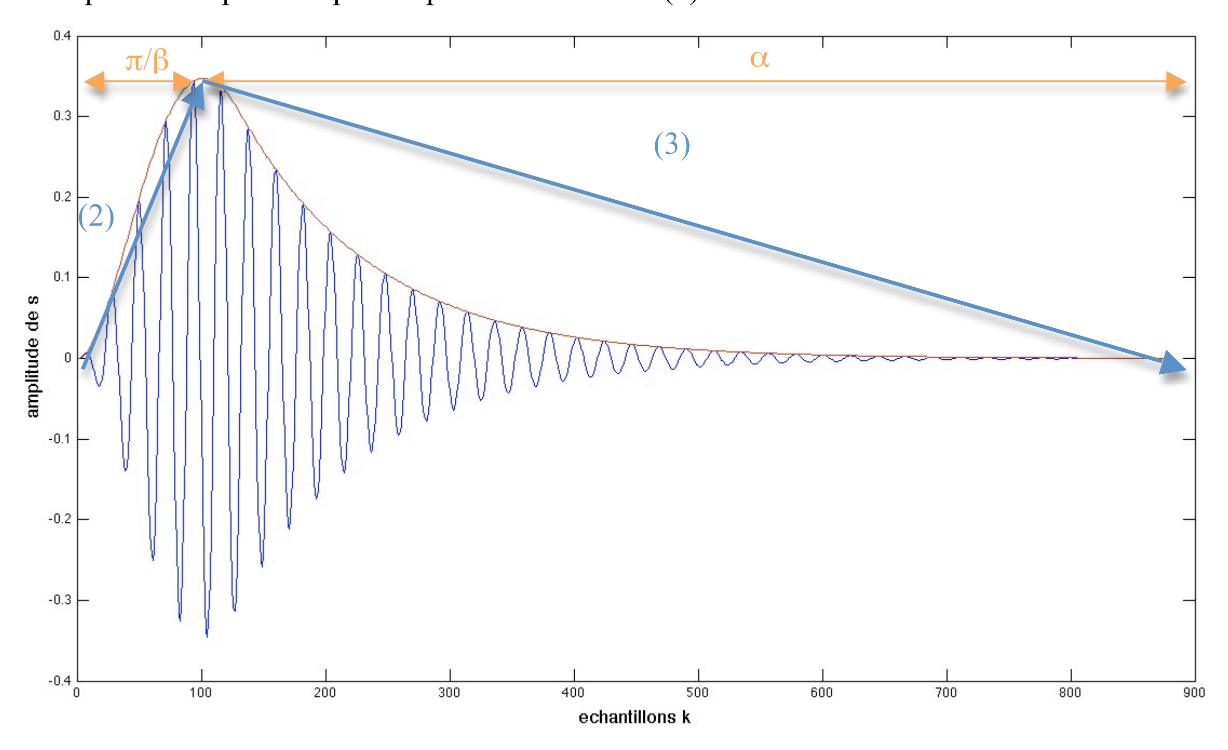


Fig. 33 – Fonction d'Onde Formantique – D'après le script Matlab 'fof.m'[87].

87 Disponible sur le cd à cd/matlab et dans l'annexe n°2.

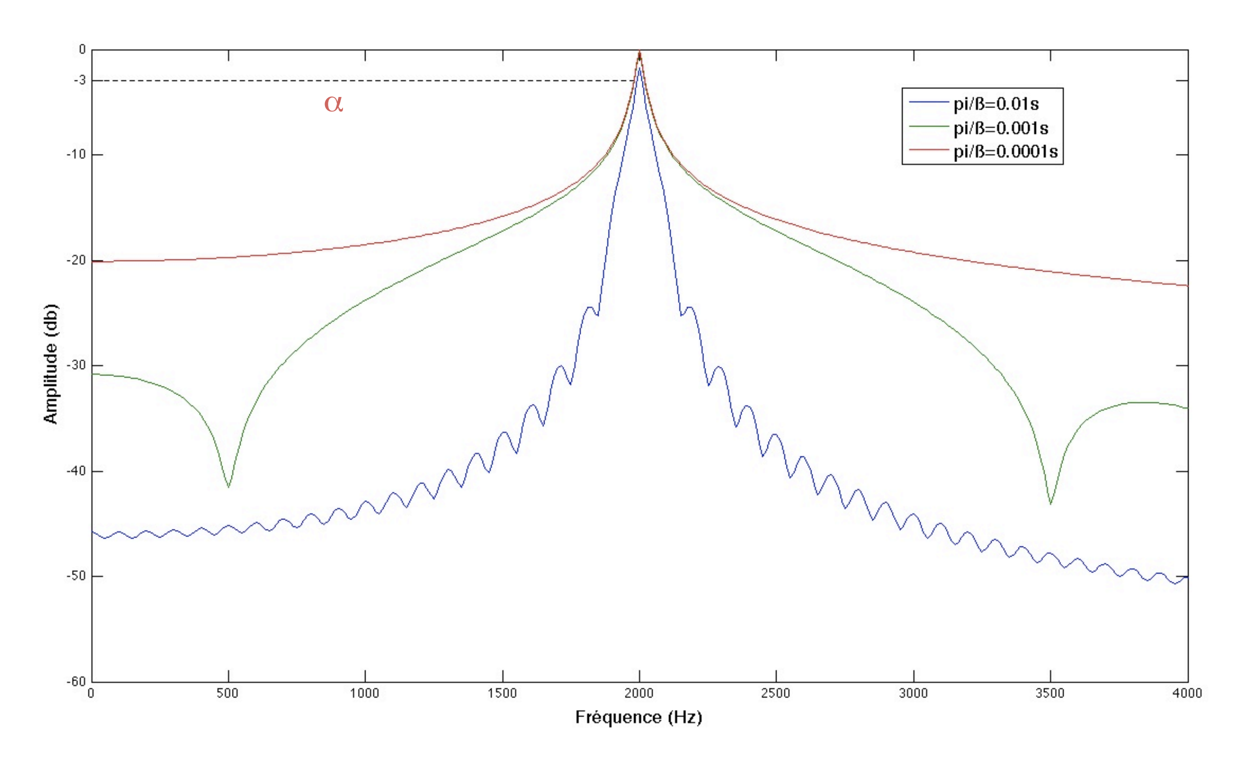


Fig. 34 – Analyse FFT de trois FOFs avec W_0=2000 Hz, α=80 Hz et des valeurs de π/β différentes – D'après le script Matlab 'fofspec.m'[88].

c) Création de FOFs dans CSOUND

CSOUND est un logiciel libre développé sous licence LGPL[89]. Il est principalement composé d'un compilateur utilisé grâce à un langage spécifique dérivé du C pour effectuer des tâches de traitement numérique du signal orientées pour la musique.[90]

L'objet `fof`[91] dans CSOUND permet de générer un flux d'impulsions du même type que celles décrites ci-dessus. Elles présentent toutefois un certain nombre de différences. En effet, il est possible de choisir le type d'enveloppe appliquée parmi toutes les tables de fonctions disponibles dans le programme. Ainsi, dans le fichier CSOUND présenté dans l'annexe n°26, l'enveloppe d'amplitude de chaque impulsion est générée par la fonction GEN19[92]. Cette fonction permet habituellement de construire une forme d'onde composée d'une sinusoïde. Dans le cas présent, elle est configurée de la manière suivante dans CSOUND :

```
f2 0 1024 19 0.5 a' ϕ' b'
```

---

88 Disponible sur le cd à cd/matlab et dans l'annexe n°3.

89 Lesser General Public Licence (Licence publique générale limitée).

90 http://www.csounds.com/ Site officiel du programme CSOUND (en ligne le 04/09/2010).

91 VERCOE, Barry, « FOF », *The canonical CSOUND Reference Manual, Version 5.09, éd. sous la direction de Barry Vercoe*, Cambridge : MIT, 2005, p. 749-751.

92 *Ibid.*, p. 2020-2021.

avec :

a'=0.5, l'amplitude de la sinusoïde

ϕ'=270, le déphasage en degré

b'=0.5, le décalage de l'amplitude pour que la fonction soit strictement positive : $0 \leq f2 \leq 1$.

Le résultat produit est alors de la forme : $x(t) = \frac{\sin(t+\phi)+b}{a}$ avec :

$\phi = \frac{2\pi 3}{4}$, le déphasage en radiant

b=1, pour que pour toute valeur de *t*, *x* soit positif

a=2, pour que pour toute valeur de *t*, $0 \leq x \leq 1$.

x(*t*) est utilisé par CSOUND pour construire l'attaque puis est ensuite lu à l'envers pour construire la chute. Le fichier MATLAB « sigmo.m »[93] permet de calculer et d'afficher cette fonction :

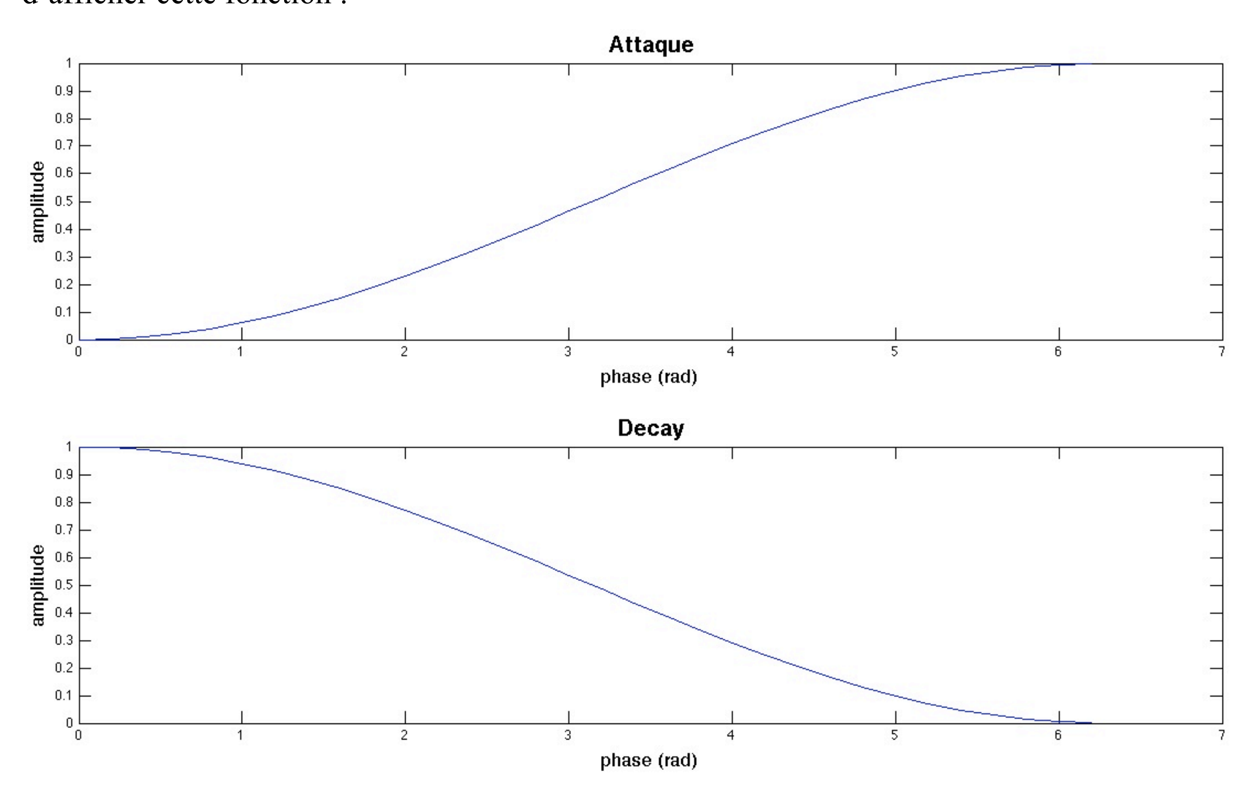


Fig. 35 – Sigmoïde générée par GEN19 utilisée pour le calcul de l'enveloppe d'amplitude des FOFs dans CSOUND – D'après le script MATLAB 'sigmo.m'[94].

L'enveloppe générée dans CSOUND avec GEN19 est pratiquement identique à celle obtenue avec les formules données précédemment comme la montre la figure 36. On

93 Disponible sur le cd à cd/matlab et dans l'annexe n°4.

94 Ibid.

peut simplement noter que l'amplitude décroit légèrement moins rapidement et que la durée de α est rendue légèrement plus courte dans CSOUND.

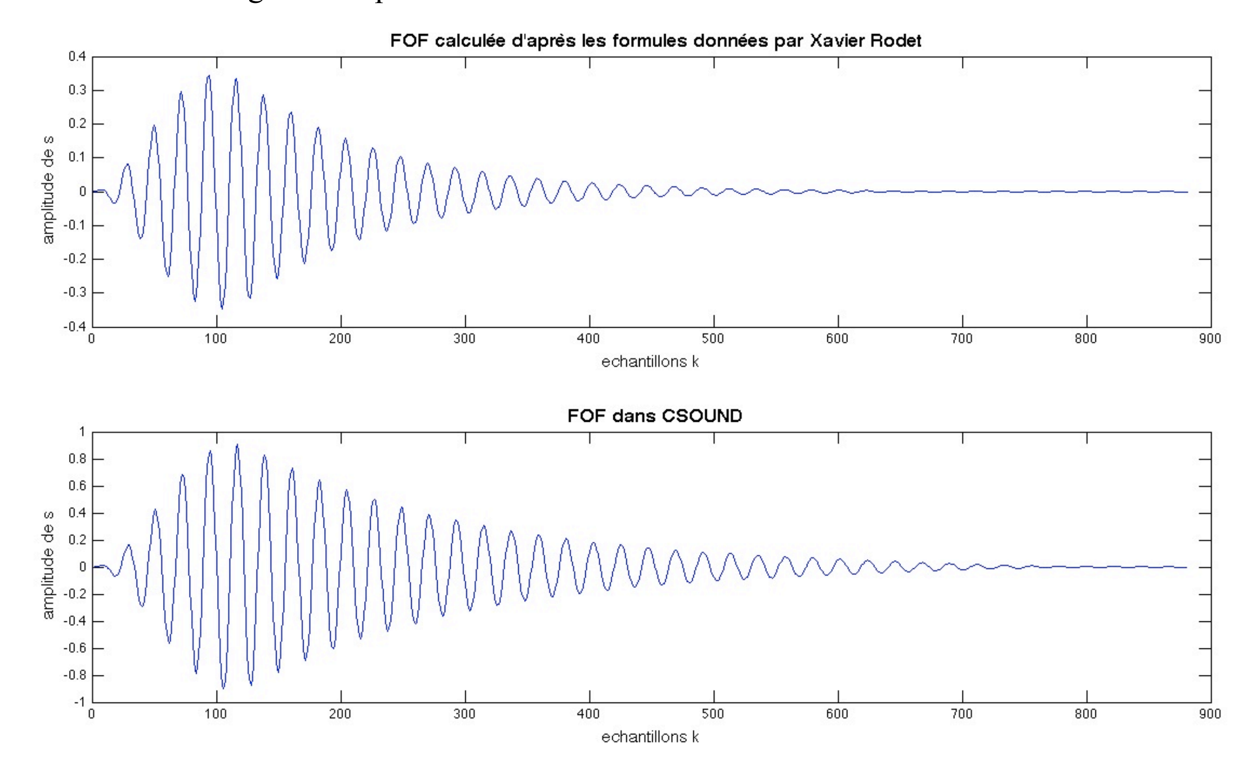


Fig. 36 – Comparaison d'une FOF générée avec les formules données par Xavie Rodet[95] avec une FOF générée dans CSOUND (fof-dem.csd et fof-dem.aiff[96]) avec l'*opcode* `fof` et GEN19 (paramètres identiques) - D'après le script Matlab 'fof.m'[97].

L'autre différence entre l'objet `fof` de CSOUND et le modèle utilisé dans CHANT concerne α. En effet, dans CHANT, α contrôle la largeur de bande du spectre des FOFs à -3db, alors que dans CSOUND, ce contrôle se fait à -6db. Ainsi, dans ce dernier cas, le paramètre α a un impact plus important sur le résultat que dans le premier puisqu'il agit sur une zone plus étendue du spectre.

L'objet `fof` de CSOUND présenté dans l'annexe n°26 permet d'effectuer l'étape de synthèse dans le système présenté dans le chapitre III.

d) Synthétiseur FOF

En appliquant les principes vus ci-dessus, il est possible de créer un synthétiseur dont la structure est la suivante :

95 RODET, Xavier, « Time-Domain Formant-Wave-Function Synthesis », *Computer Music Journal*, VIII (1984), n° 3, p. 9-14.

96 Disponible sur le cd à cd/csound.

97 Disponible sur le cd à cd/matlab et dans l'annexe n°2.

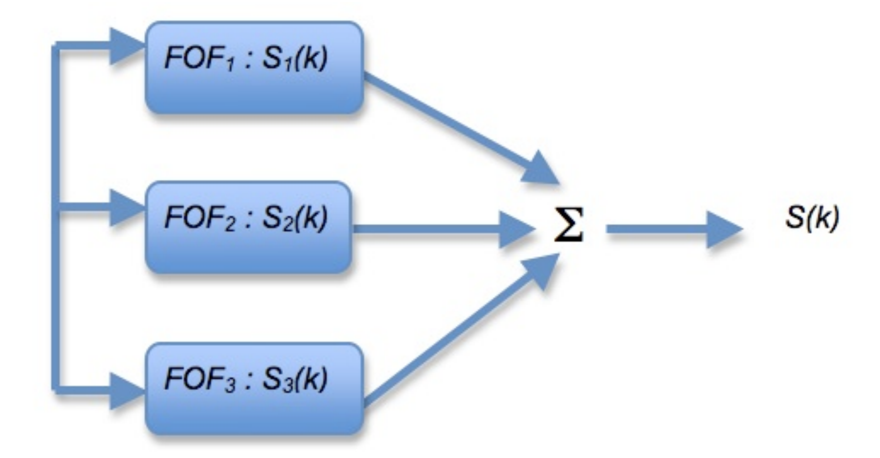


Fig. 37 – Structure d'un synthétiseur FOF.

Les paramètres pour chaque période fondamentale sont donc :

- Les $2\pi w_i$ fréquences centrales
- Les BW_i largeurs de bandes
- Les A_i amplitudes
- Les π/β_i largeurs de jupes
- Les ϕ_i phases initiales.

En plus de ses nombreux avantages techniques, la synthèse par Fonction d'Onde Formantique permet de reproduire des comportements vocaux très complexes à implémenter avec un modèle source/filtre. Toutefois, il n'est possible de synthétiser que des sons de type voisé comme les voyelles[98] avec un synthétiseur tel que celui présenté dans l'annexe n°26. Ainsi, pour être complet, ce modèle doit également posséder un système capable de produire des sons vocaux de type bruité comme les consonnes. Pour ceci, il est possible d'utiliser une source de bruit blanc passée dans un banc de filtres dynamiques. Ainsi, si on complète la figure 37, on obtient :

---

98 ALLESSANDRO, Christophe (d') ; RODET, Xavier, « Synthèse et analyse-synthèse par fonctions d'ondes formantiques », *Journal* d'*Acoustique*, n°2 (1989), p. 163-169.

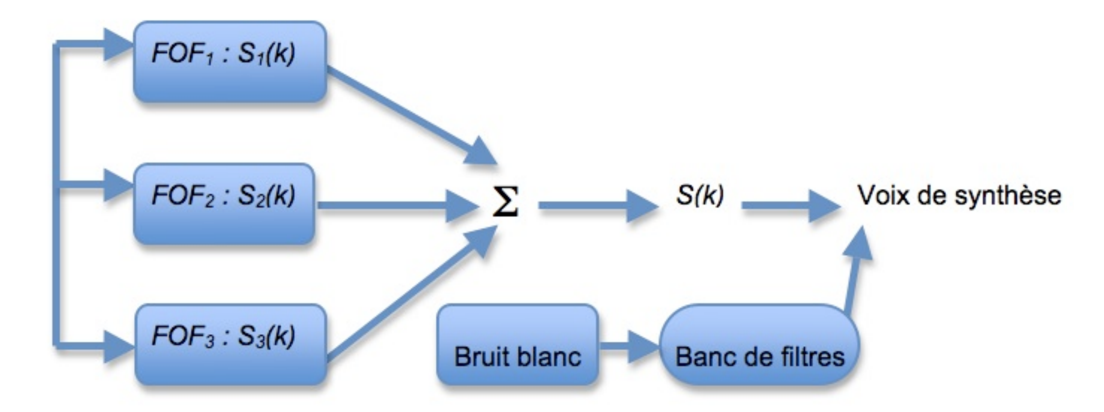


Fig. 38 – Structure d'un synthétiseur FOF complet.

Ce modèle est le même que celui utilisé dans CHANT. Il sera implémenté dans CSOUND dans le chapitre III.

e) Avantages de la synthèse par FOFs face aux modèles de synthèse par filtres

On sait d'après le chapitre (I)-(A)-(b) que le conduit vocal produit des sons dont les fréquences des formants et de la fondamentale f_0 évoluent en permanence. Ces variations, bien qu'elles soient infimes (de l'ordre de + ou – 2%) sont primordiales pour la perception d'un son vocal. Les coefficients d'un filtre ne pouvant pas subir de discontinuités, il est nécessaire de les interpoler entre chaque nouvelle valeur, opération très coûteuse en calculs.

Les FOFs, sont quant à elles par construction non discontinues. En effet, comme on l'a montré précédemment, le signal S issu d'un générateur de FOFs est la somme des réponses $S_n(k)$ décalées d'une période du fondamental $T=1/F_0$ où n indexe les impulsions successives. Ainsi, les FOFs acceptent des valeurs différentes à chaque période.

Il arrive que certaines composantes du spectre issues d'un synthétiseur FOF se trouvent en opposition de phase et donc s'annulent. Ceci à pour effet l'apparition de « trous » au niveau des fréquences centrales de deux cellules voisines. Une méthode simple pour remédier à ce problème consiste à déphaser la réponse d'une cellule par rapport à une autre en changeant les valeurs de ϕ. Une telle opération pourrait être implémentée avec des filtres, toutefois, cette tâche serait particulièrement ardue à mettre en place car soit inadéquate, soit trop complexe.

Un autre avantage du modèle par FOF est la richesse spectrale dans le cas de l'utilisation d'une largeur de bande *BW* réduite et d'une fluctuation rapide de *w*. Ceci est dû au fait que les FOFs déclenchées à chaque période *n* sont pratiquement des sinusoïdes de fréquences fixes différentes les unes des autres, alors que la réponse d'une cellule d'un filtre dans de telles conditions ressemble à une sinusoïde dont la fréquence centrale évolue continûment.

B. Le programme CHANT[99] [100]

La première implémentation de la technique de synthèse par fonctions d'onde formantique fut le programme CHANT, développé par l'équipe de Xavier Rodet à l'IRCAM à partir de 1978. Cet outil de synthèse par règles « a été conçu comme un instrument de composition perceptuelle pour traiter ensemble production et perception, matériau et organisation »[101].

Les différentes règles qui agissent sur les paramètres du programme permettent un contrôle de l'expressivité, de la qualité du timbre, des intonations et des nuances de la voix chantée « dépassant la durée de la note pour s'articuler autour de phrases musicales »[102].

CHANT a été utilisé dans la création d'un certain nombre d'œuvres dans les années quatre-vingt et au début des années quatre-vingt-dix telles que *Mortuos Plango, Vivos Voco* de Jonathan Harvey en 1980[103], *Les chants de l'amour* de Gérard Grisey (1985)[104] ou encore *Chréode* de Jean-Baptiste Barrière (1983)[105]. Depuis sa création en 1979, CHANT a connu beaucoup de modifications et a été porté sur un grand nombre de systèmes. Bien que ce programme reste l'un des outils majeur pour la synthèse de la voix chantée, il est aujourd'hui quelque peu laissé à l'abandon.

A l'heure actuelle, l'utilisation et le contrôle de CHANT se fait par l'intermédiaire d'autres logiciels tels que Max/MSP ou Diphone Studio.

a) Historique du programme CHANT[106] [107]

99 BAISNEE, Pierre-François*, CHANT manual*, Document Ircam : Paris, 1985.

100 POTTIER, Laurent*, Le contrôle de la synthèse sonore, le cas particulier du programme PatchWork*, Thèse de doctorat (inédite), Paris : EHESS, 2001, p. 68-80.

101 RODET, Xavier ; BARRIERE, Jean-Baptiste ; POTARD, Yves, *Rapport de recherche n°35 : Chant, de la synthèse de la voix à la synthèse en général*, Paris : IRCAM, 1985, p. 16.

102 POTTIER, Laurent, *op. cit.*, p. 68.

103 BOSSIS, Bruno*, La voix et la machine : La vocalité artificielle dans la musique contemporaine*, Rennes : Presses Universitaire de Rennes, 2005, p. 132-146.

104 *Ibid*., p. 146-172.

105 http://brahms.ircam.fr/works/work/6632/ (en ligne le 04/09/2010).

106 Laurent Pottier, *op. cit.*, p.68.

107 BOSSIS, Bruno, *La voix et la machine : La vocalité artificielle dans la musique contemporaine*, Rennes : Presses Universitaire de Rennes, 2005, p. 129-131.

Le point de départ des recherches qui ont donné naissance au programme CHANT est le système de synthèse vocale SARA développé par Xavier Rodet au centre CEA[108] de Saclay. La première version de CHANT a été écrite en SAIL[109] sur Dec-PDP10[110] en 1979 puis a rapidement été traduite en Fortran par Jean Holleville en 1981. Cette même année, CHANT est installé sur la Samson-Box du CCRMA[111] au SAIL de Stanford. Ce système pouvait alors calculer environ deux cents FOFs simultanément procurant une synthèse de bonne qualité. Toujours au début des années quatre-vingt, CHANT a été codé pour fonctionner sous UNIX par Yves Potard et Jan Vandenheede sur un ordinateur Dec-VAX780 au centre EMS[112] de Stockholm.

Dans ses premières versions, le programme CHANT n'intégrait pas de banc de filtres, en effet, cet élément ne fut ajouté au programme qu'en 1984. Un étage d'interpolation linéaire entre l'entrée des paramètres de contrôle et le synthétiseur FOF a aussi été inséré afin d'économiser les flux des données de contrôle.

A partir de 1986, le contrôle de CHANT est devenu possible par l'intermédiaire d'un environnement plus convivial élaboré sous Lisp : le programme Formes. « Ce système interactif, comprenait un langage de programmation « orienté processus », des algorithmes servant à l'aide à la composition et à la synthèse ainsi que des bibliothèques d'exemples »[113]. Le compositeur était libéré des difficultés d'accès et de compréhension des modèles de synthèse.

En 1989, CHANT a été réécrit en C sous UNIX par Francisco Iovino. Cette modification lui a permis d'être implémenté sur des machines Macintosh où il a pu bénéficier peu de temps après du contrôle par PatchWork. Il devint alors polytimbral et polyphonique et pouvait être contrôlé par Formes, Scheme et Common Lisp.

CHANT a aussi été implémenté sur de nombreuses plateformes pour être contrôlé en temps réel. Ainsi, en 1981, Xavier Rodet et Yves Potard l'ont fait fonctionner sur la machine 4C. En 1987 également, il a été implémenté sur un processeur Mercury en périphérique UNIX sur un ordinateur SUN par Xavier Rodet et Gerhard Eckel.

108 Commissariat à l'Energie Atomique.

109 Stanford Artificial Intelligence Laboratory.

110 Ordinateur central « mainframe » développé par la Digital Equipment Corporation (DEC) à partir de la fin des années soixante.

111 Center for Computer Research in Music and Acoustics.

112 Electroacoustic Music in Sweden.

113 POTTIER, Laurent, *Le contrôle de la synthèse sonore, le cas particulier du programme PatchWork*, Thèse de doctorat (inédite), Paris : EHESS, 2001, p. 68.

A l'heure actuelle, CHANT perdure encore dans quelques programmes développés à l'IRCAM. En effet, depuis 1999, il est intégré au logiciel Diphone Studio. Il fut également porté à la même époque dans OpenMusic et dans Max/MSP.

b) Modèle de production de CHANT

La spécificité de CHANT est son utilisation de la synthèse par fonctions d'onde formantique. En plus de celle-ci, CHANT emploie deux autres techniques pour la synthèse de sons vocaux. En effet, comme nous l'avons indiqué précédemment, les FOFs ne permettent pas la reproduction de sons de type fricatif tels que les consonnes. C'est une des raisons pour lesquelles CHANT est pourvu d'un générateur de bruit blanc et d'un banc de filtres. Ce dernier peut aussi être appliqué sur des sons extérieurs.

Ces différents éléments peuvent alors être connectés entre eux selon plusieurs modèles en fonction du résultat désiré comme le montre la figure 39.



Fig. 39 – Différents modèles de connexion des FOFs, filtres, fichiers sons et bruits dans le synthétiseur CHANT de Diphone Studio.

C. Les règles de synthèse de CHANT

La version de base de CHANT compte un grand nombre de paramètres. Ainsi, afin de simplifier son utilisation, ces paramètres peuvent être contrôlés de façon automatique grâce à des règles et disposent de valeurs par défaut établies d'après des mesures et des analyses effectuées sur la voix. Ces valeurs sont stockées dans une base de données et sont classées par voyelles et par types de voix (généralement soprano, alto, contreténor, ténor et basse). Selon Pierre-François Baisnée, CHANT est donc « le synthétiseur le plus simple à manier qu'il soit »[114].

Les règles de CHANT rendent possible la synthèse de phrases chantées avec un nombre très réduit de paramètres tels que le type de voix désiré et la partition, décrite par des hauteurs, des durées et des intensités. L'utilisateur peut aussi, naturellement, modifier manuellement l'intégralité des paramètres lui donnant alors un contrôle très précis sur le résultat.

Il est possible de discerner deux types de règles dans CHANT : les règles automatiques qui calculent les paramètres des formants en fonction de la partition fournie et les autres algorithmes qui concernent le vibrato, les microvibrations du son et les enchaînements de notes.

Afin de comprendre pleinement la réimplantation du programme CHANT présentée dans le chapitre III, il est primordial d'étudier en détail le fonctionnement des différentes règles et fonctions de CHANT. Il sera parfois fait référence au code source des règles de la bibliothèque *PW-Chant* pour PatchWork[115].

a) Contrôle du vibrato[116]

Le vibrato est un facteur très important dans la perception de la « beauté » vocale dans le chant occidental[117]. De plus, il permet de renforcer l'identité des formants en créant un balayage dans le spectre de la voix tel qu'il l'a été montré dans (I)-(C)-(a).

---

114 BAISNEE, Pierre-François, *op. cit.*

115 IOVINO, Francisco ; LAURSON, Mikael, *fichier voice-rules.Lisp de la bibliothèque PW-Chant*, Paris : IRCAM, 1996, disponible sur le cd à cd/PW et dans l'annexe n°4.

116 BAISNEE, Pierre-François, *op. cit.*

117 RODET, Xavier ; BARRIERE, Jean-Baptiste ; POTARD, Yves, *Rapport de recherche n°35 : Chant, de la synthèse de la voix à la synthèse en général*, Paris : IRCAM, 1985, p. 17.

Dans CHANT, le vibrato est créé par un oscillateur générant un signal périodique de basse fréquence qui permet de moduler la fréquence f_0 des différents générateurs de FOF. Typiquement, la fréquence d'un vibrato vocal varie entre 5 et 7 Hz tandis que les modulations de la fréquence de la fondamentale sont de l'ordre de +/- 1,1% à +/- 3,7% chez les femmes et de +/- 2% à +/- 4% chez les hommes. L'analyse du spectre de deux sons de synthèse FOF avec cinq formants produits par l'objet CSOUND « fof-sop.csd »[118] est présentée dans la figure 40. On peut y voir l'effet de l'introduction d'un vibrato sur la fondamentale et ses différents harmoniques.

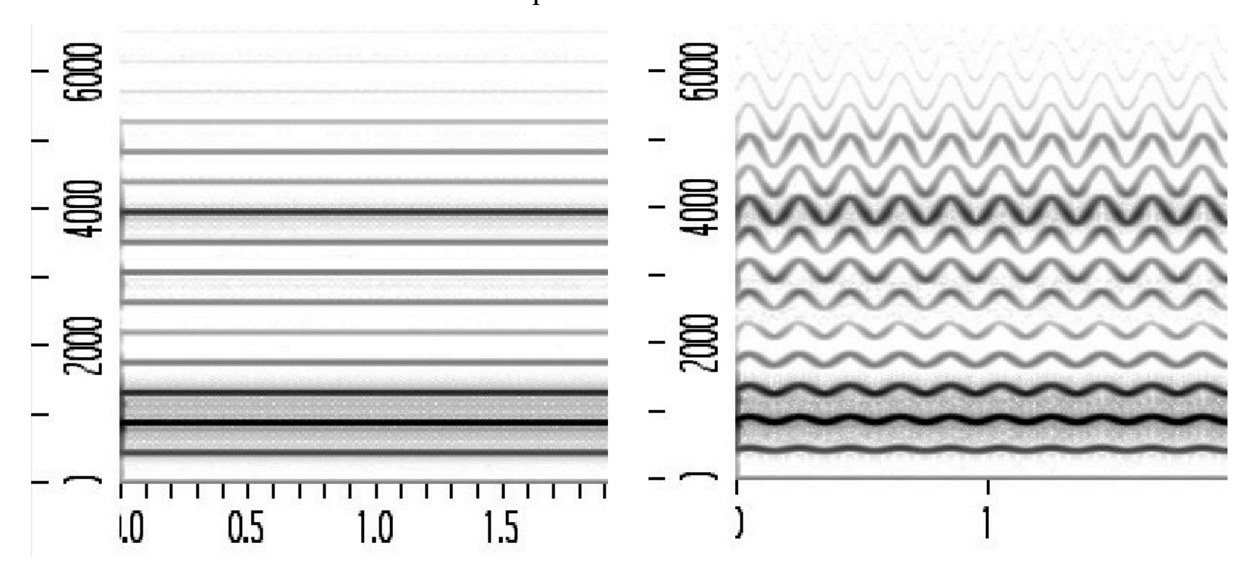


Fig. 40 – Sonagramme de deux sons produits par l'objet CSOUND « fof-sop.csd », à gauche sans vibrato (fof-sop.aiff[119]), à droite avec vibrato (fof-sop-vib.aiff[120]).

Afin que le vibrato produit sonne le plus naturel possible, il est nécessaire d'introduire des microvariations aléatoires au niveau de sa fréquence et de son amplitude. En effet, les cordes vocales ne sont pas en mesure de moduler de façon périodique leur fréquence de vibration de manière aussi précise qu'un oscillateur. Ces micromodulations sont implémentées dans CHANT grâce à un ensemble de quatre jitters qui permettent de générer des valeurs aléatoires à une fréquence donnée. Ainsi, la fréquence et l'amplitude du vibrato sont modulées de la façon suivante dans CHANT :

Fréquence finale = (*fréquence du vibrato*)*((*jitter1*+*jitter2*)/2)

Amplitude finale = (*amplitude du vibrato*)*((*jitter3*+*jitter4*)/2)

Les amplitudes de *jitter1* et *jitter2* indiquent donc le taux de variations de la fréquence du vibrato, celles de *jitter3* et *jitter4* le taux de variations de l'amplitude du vibrato.

---

118 Disponible sur le cd à cd/csound et dans l'annexe n°5.

119 Disponible sur le cd à cd/csound.

120 Ibid.

b) Variations aléatoires de la fondamentale[121]

Le son produit par les cordes vocales n'a pas une fréquence totalement fixe. En effet, tout comme pour le vibrato présenté précédemment, on peut observer des microfluctuations de la fréquence f_0 de la fondamentale. Ces variations infimes, mais néanmoins très importantes pour restituer l'aspect naturel du son de la voix, sont implémentées dans CHANT grâce à un ensemble de trois jitters connectés en parallèle et fonctionnant de la même manière que ceux utilisés dans la modulation de la fréquence et de l'amplitude du vibrato décrite précédemment.

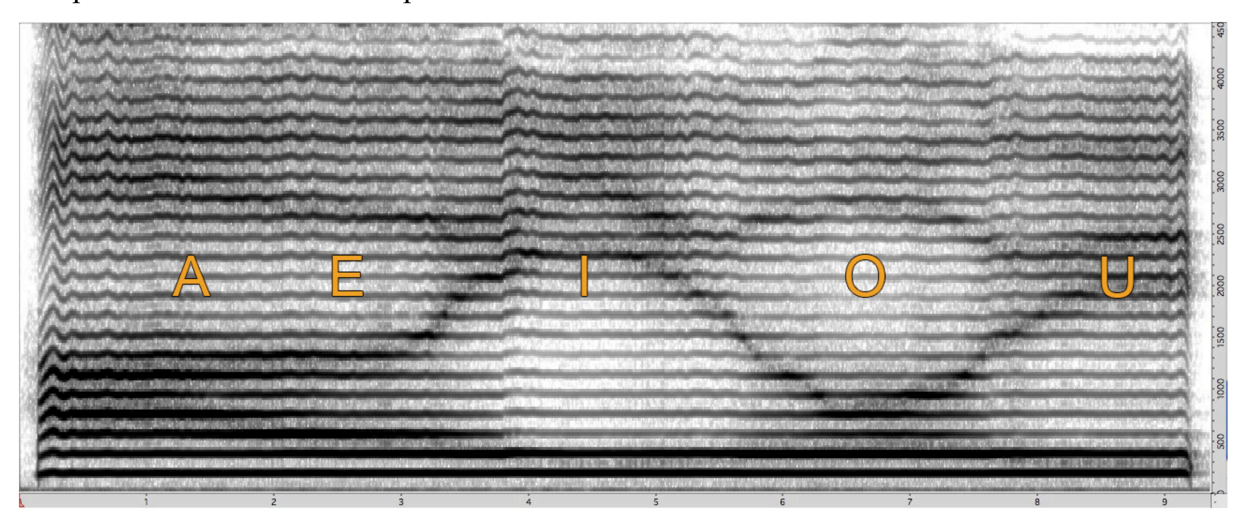


Fig. 41 – Transitions entre plusieurs voyelles par interpolation des fréquences des formants (transitions-voyelles.aiff[122]).

c) Interpolation des formants et de la fondamentale[123]

○ Interpolation des fréquences f_0 de la fondamentale

Un effet récurant utilisé dans la pratique du chant est l'introduction de glissandos entre deux notes. Ils sont produits en changeant progressivement la fréquence de vibration des cordes vocales d'une note à une autre. D'un point de vue physique, cette opération revient à faire une interpolation linéaire entre les fréquences f_0 de ces deux notes comme cela est montré dans la figure suivante :

---

121 BAISNEE, Pierre-François, *CHANT manual*, Document Ircam : Paris, 1985, p. 41.

122 Disponible sur le cd à cd/audio.

123 POTTIER, Laurent, *Le contrôle de la synthèse sonore, le cas particulier du programme PatchWork*, Thèse de doctorat (inédite), Paris : EHESS, 2001, p. 75.

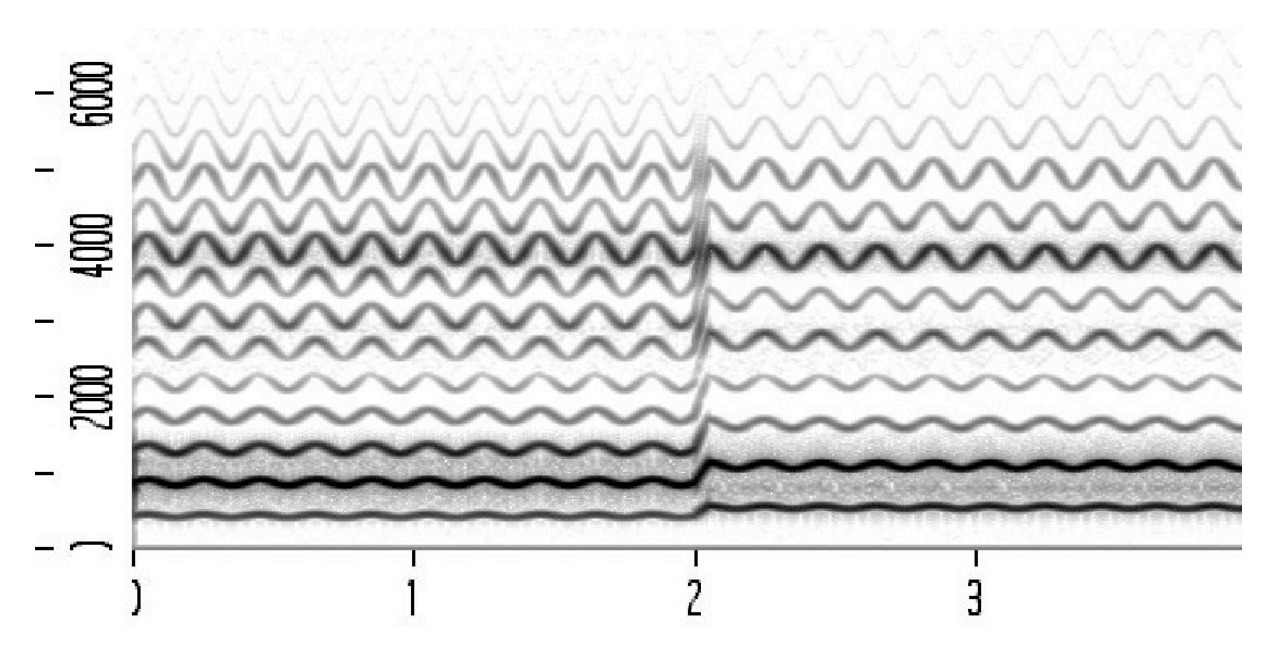


Fig. 42 – Sonagramme d'un son produit par l'objet CSOUND « fof-sop.csd » montrant une interpolation entre deux notes (à $2s^{-1}$).

Les harmoniques se trouvant au dessus de la fondamentale étant des résonnances de celle-ci, ils sont eux aussi interpolés. La durée de l'interpolation définie la durée du glissando. Dans CHANT, il est possible d'activer ou de désactiver l'interpolation des fréquences de la fondamentale en fonction du contexte de la synthèse. Par exemple, l'interpolation devra être activée avec une durée très courte dans le cas d'un legato alors qu'il sera nécessaire de l'empêcher dans le cas de notes piquées.

- **Interpolation des voyelles**

Lors du passage d'une voyelle à une autre dans une phrase legato ou dans un mot, les paramètres des différents formants changent de façon progressive. L'interpolation des différentes fréquences, amplitudes et largeurs de bandes des formants permet d'effectuer ces transitions dans CHANT. Tout comme pour l'interpolation des fréquences de la fondamentale, il n'est pas nécessaire d'activer l'interpolation des formants dans le cas de sons piqués.

d) Contrôle de l'enveloppe globale des notes

Il est possible de contrôler de façon indépendante l'enveloppe de chaque note dans CHANT. Il est toutefois important de préciser que les différentes versions de CHANT ont vu se succéder un certain nombre de générateur d'enveloppes donnant des résultats légèrement différents.

Dans la version VAX-780 de 1985, l'enveloppe était définie par trois paramètres : la durée totale de l'enveloppe et les proportions de la durée de l'attaque et de la chute sur la durée totale de l'enveloppe. Ces trois phases étaient totalement linéaires. Enfin, il est important de préciser que si la durée totale de l'enveloppe était inférieure à celle de la note, un silence correspondant à la différence entre ces deux valeurs était introduit.[124] La figure 43 montre la forme d'une enveloppe typique de celles proposées sur le système VAX-780 de CHANT :

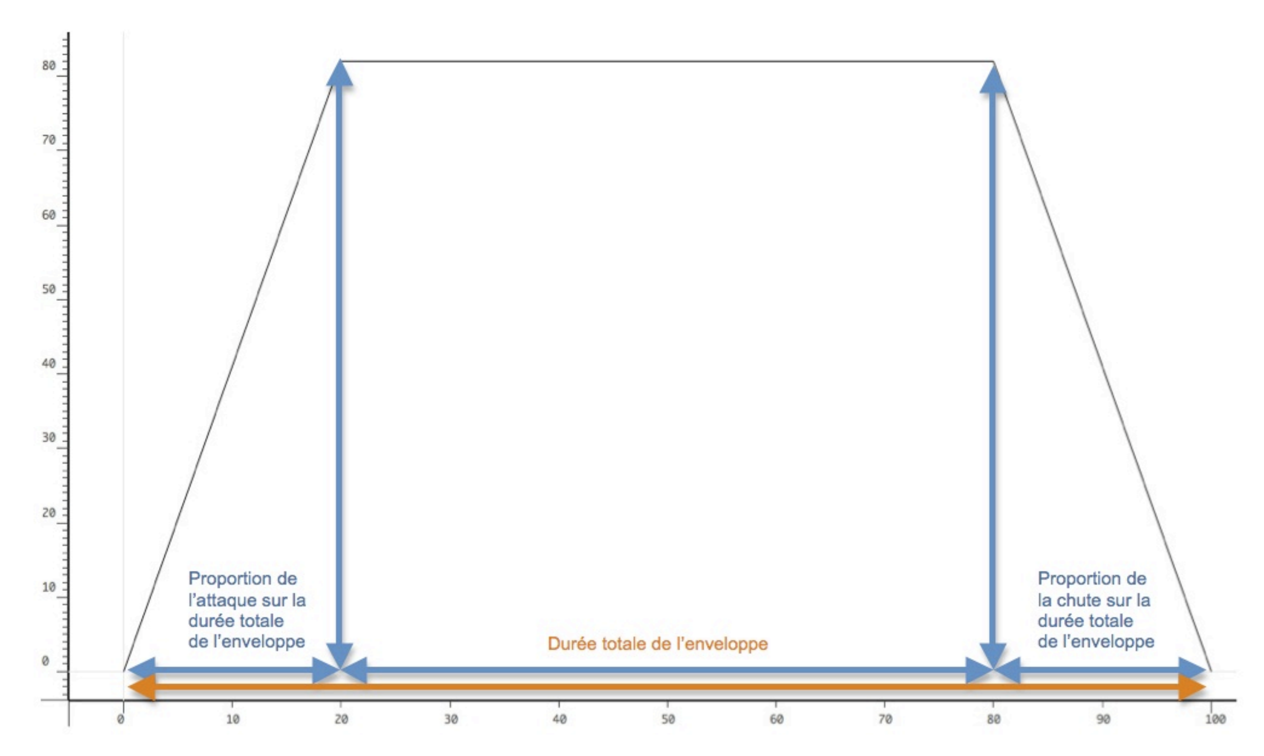


Fig. 43 – Forme de l'enveloppe globale d'une note dans la version VAX-780 de CHANT.

La version 1.2 de CHANT pour Patchwork de 1996 dispose d'un générateur d'enveloppe légèrement différent de celui présenté précédemment. En effet, chacune des parties de l'enveloppe (attaque, maintien et chute) est produite indépendamment par une portion de fonction sinusoïdale.[125] Il en résulte une enveloppe avec des formes plus arrondies comme cela est montré dans la figure suivante :

---

124 BAISNEE, Pierre-François, *CHANT manual*, Document Ircam : Paris, 1985, p. 6.

125 IOVINO, Francisco ; LAURSON, Mikael, « envelope », *PatchWork PW-Chant reference*, Paris : IRCAM, 1996, p. 51.

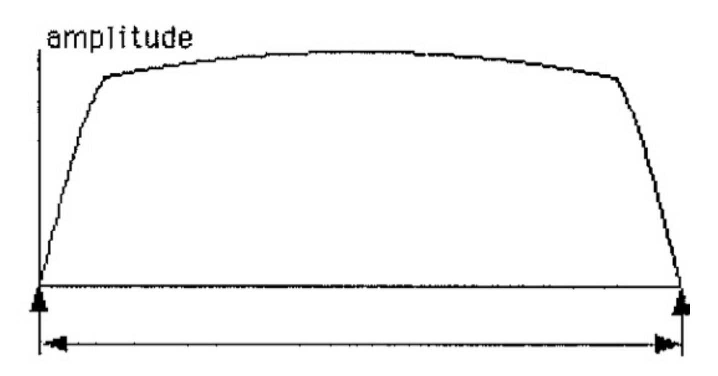


Fig. 44 – Exemple d'enveloppe produite par le module *envelope* du programme Patchwork.[126]

e) Règle pour le calcul automatique des largeurs de bande des formants

Chant dispose d'une règle capable de calculer la largeur de bande d'un formant à partir de sa fréquence. D'après Pierre-François Baisnée, les largeurs de bande des formants varient en fonction de leur fréquence centrale suivant une courbe parabolique définie sur le spectre par trois points. Ces trois points sont calculés à partir de trois fréquences et de trois largeurs de bande de référence selon l'algorithme suivant[127] :

Fréquences de référence = ($fref_0$ $fref_1$ $fref_2$)

Largeurs de bande de référence = ($bwref_0$ $bwref_1$ $bwref_2$)

$v = fref_0^2 - fref_1^2$

$u = fref_0 - fref_2$

$w = bwref_0 - bwref_1$

$x = fref_0 - fref_1$

$y = fref_1 - fref_2$

$z = x/y$

$d = v - ((fref_1^2 - fref_2^2).z)$

Si x, u et d sont différents de 0, alors les points de référence pour la parabole peuvent être calculés de la manière suivante :

$ptref2 = (z.(w - bwref_1 - bwref_2))/d$

$ptref1 = (w - (ptref2.v))/x$

$ptref0 = (fref_0.ptref1) + (fref_0^2.ptref2) - bwref_0$

Les fréquences et les largeurs de bande de référence peuvent être définies par l'utilisateur. Elles disposent néanmoins de valeurs par défaut qui donnent des résultats de très

---

126 POTTIER, Laurent*, Le contrôle de la synthèse sonore, le cas particulier du programme PatchWork*, Thèse de doctorat (inédite), Paris : EHESS, 2001, p. 75.

127 IOVINO, Francisco ; LAURSON, Mikael, « Rule for the Automatic Calculation of the Bandwidths », *fichier voice-rules.lisp de la bibliothèque PW-Chant*, *op. cit.*

bonne qualité dans la plupart des cas : $bw_0 = 75$, $bw_1 = 75$, $bw_2 = 150$, $fref_0 = 200$, $fref_1 = 500$ et $fref_2 = 4000$. Dans ce cas : $ptref0 = 471$, $ptref1 = 138{,}6$ et $ptref2 = 12{,}04$.

La fonction utilisée pour le calcul des largeurs de bande des formants en fonction de leur fréquence est de la forme suivante :

$$bw_i = ptref0 - ptref1Ln(freq_i) + ptref2Ln(freq_i)^2$$

En utilisant les fréquences et largeurs de bande par défaut, on obtient donc la fonction suivante dont il est possible de voir la forme sur la figure 45 :

$$bw_i = 471 - 138{,}6Ln(freq_i) + 12{,}04Ln(freq_i)^2$$

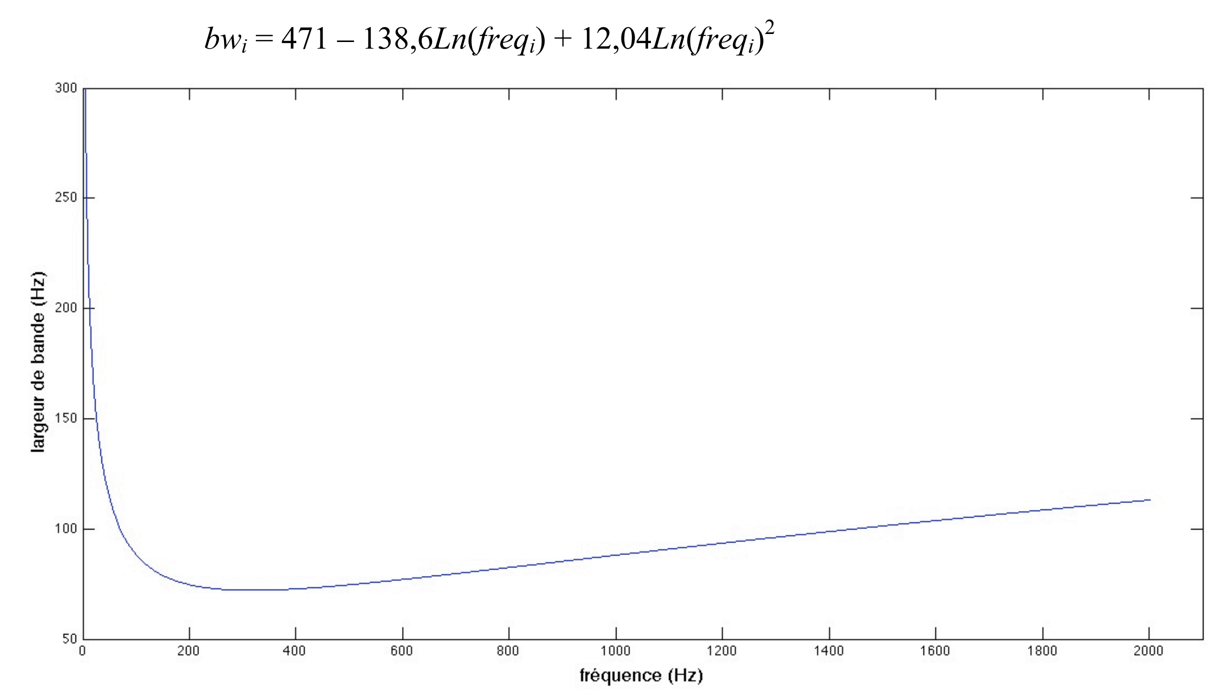


Fig. 45 – Calcul de la largeur de bande des formants en fonction de leur fréquence dans CHANT en utilisant les fréquences et largeurs de bande de référence par défaut, d'après le script MATLAB « chantbw.m[128] ».

f) Règles pour le calcul automatique et l'ajustement des amplitudes des formants

Les amplitudes des différents formants jouent un rôle primordial dans la détermination du timbre. Leurs valeurs peuvent êtres amenées à varier en fonction d'un grand nombre de facteurs.

---

128 Disponible sur le cd à cd/matlab et dans l'annexe n°6.

○ Calcul automatique des amplitudes des formants[129] [130]

Le calcul automatique des amplitudes des formants se fait dans CHANT grâce à une règle simulant une série de filtres qui en fonction de la fréquence de chacun d'entre eux, donne les amplitudes des formants. Le but de ce calcul est de prendre en compte les différentes interactions possibles entre les formants. Par exemple, lorsque deux formants s'approchent l'un de l'autre, la simulation de filtres permet le renforcement de leurs amplitudes exactement de la même façon que dans le système phonatoire.

Un autre effet de cette règle est de calculer l'amplitude des formants selon une loi inversement proportionnelle à leur fréquence. Ainsi, l'enveloppe spectrale résultante n'est pas plate mais a une forme descendante.

○ Ajustement des amplitudes des formants[131] [132]

Il a été démontré dans (I)-(C)-(a) que l'amplitude des formants de la voix varie en fonction de l'effort vocal fait par un chanteur. En effet, il ne suffit pas d'augmenter ou de réduire de façon linéaire l'amplitude des formants pour modifier les nuances d'une phrase musicale.

Le programme CHANT dispose d'une règle permettant d'ajuster l'amplitudes des différents formants du son synthétisé en fonction du type de voix utilisé (soprano, alto, contreténor, etc.). Les paramètres nécessaires à son fonctionnement sont l'indice d'inclinaison du spectre (*cslope*), le type de voix (*sex*), la fréquence médiane de la tessiture de la voix utilisée (f_0*moyen*), la liste des amplitudes des formants à ajuster, la fréquence de la fondamentale (f_0) et un coefficient dynamique pour la modification du spectre (*coefamp*). L'algorithme de la règle d'ajustement des amplitudes de formants peut être résumé de la façon suivante :

```
tino = coefamp.(f0/f0moyen)^ajus3
Si cslope est négatif, alors les fonctions suivantes sont utilisées :
      Dans le cas d'une voix masculine :
      amplitude ajustée = amplitude.tino.coefamp.(3+(1,1.((400-f0)/300)))
      Dans le cas d'une voix féminine :
```

129 BAISNEE, Pierre-François, *CHANT manual*, Document Ircam : Paris, 1985, p. 34.

130 IOVINO, Francisco ; LAURSON, Mikael, « Rule for the Automatic Calculation of the Amplitude », *fichier voice-rules.lisp de la bibliothèque PW-Chant*, *op. cit*.

131 BAISNEE, Pierre-François, CHANT manual, Document Ircam : Paris, 1985, p. 34-36.

132 IOVINO, Francisco ; LAURSON, Mikael, « Rule for the Vocal Effort Correction », *fichier voice-rules.lisp de la bibliothèque PW-Chant*, *op. cit*.

```
amplitude ajustée = amplitude.tino.coefamp.(0,8+1,05.((1000-f0)/1250))
Si cslope est positif, il est utilisé de la manière suivante pour mettre à l'échelle les amplitudes de chaque formant :
scaler = cslope.exp(ajus1.atan(ajus2.Ln(f0/f0moyen)))
amplitude ajustée = amplitude.tino.scaler
```

g) Règle pour l'ajustement des fréquences du premier et du deuxième formant[133] [134]

Johan Sundberg a montré que dans le cas de notes situées dans le registre aigu, les fréquences des deux premiers formants sont modifiées, suivant alors les mouvements de la fondamentale f_0 lorsque celle-ci monte et pourrait dépasser la fréquence du premier formant[135].

La règle d'auto ajustement des fréquences du premier et du deuxième formant de CHANT applique les observations faites par Sundberg. Ses différents paramètres sont la liste des fréquences des formants à ajuster, la fréquence de la fondamentale (f_0), le type de voix (*sex*) et un indice de correction (*corr*). L'algorithme de cette règle peut être résumé de la façon suivante :

```
Si f0 est plus grand que la fréquence de premier formant, celle-ci est ajustée de la façon suivante :
f1 = corr.(f0-f1)
A moins que le type de voix utilisée soit celle d'un contreténor, les opérations suivantes sont effectuées :
Si f2 est plus grand ou égal à 1300 et si f0 est plus grand ou égal 200 alors :
f2 = corr.(2/3).(f0-200).((f2 – 1300)/700)
Si f2 est plus petit ou égal à 30+2f0 alors :
f2 = corr.((30+2f0)-f2)
```

133 BAISNEE, Pierre-François, CHANT manual, Document Ircam : Paris, 1985, p. 36.

134 IOVINO, Francisco ; LAURSON, Mikael, « Rule for the Automatic Bending of the First two Formants », *fichier voice-rules.lisp de la bibliothèque PW-Chant*, *op. cit*.

135 SUNDBERG, Johan, *The Science of the Singing Voice*, Dekalb (Illinois) : Northern Illinois University Press, 1987.

D. Le programme CHANT aujourd'hui : utilisation à travers Diphone Studio et Max/MSP

La synthèse par fonctions d'onde formantique ne fut pratiquement implémentée qu'à travers le programme CHANT. Comme cela a été montré dans (I)-(B)-(a), CHANT connu un important succès dans les années quatre-vingt et au début des années quatre-vingt-dix mais fut rapidement oublié et abandonné au profit d'autres programmes et d'autres techniques de synthèse. La dernière implémentation complète de CHANT se fit en 1992 dans le programme PatchWork avec la bibliothèque *PW-Chant*[136]. PatchWork fut toutefois remplacé par OpenMusic à partir de 1998 mettant définitivement fin à la dernière version de CHANT.

Malgré tout, CHANT, ou plutôt la synthèse par fonctions d'onde formantique perdure toujours de nos jours dans certains programmes développés par l'IRCAM bien que l'on reste très loin des versions complètes disponibles avant 1995. Ces différents cas vont être ici présentés.

a) CHANT dans Diphone Studio[137]

Diphone Studio est un groupe d'applications permettant l'analyse puis la synthèse de sons en utilisant différents modèles et ce de façon très intuitive. Deux types d'analyse sont disponibles dans Diphone Studio : l'analyse additive effectuée par le programme « Addan » et l'analyse par modèle de résonnance effectuée par le programme « Resan ». Les résultats de chacune d'entre-elles peuvent être ensuite utilisés pour la synthèse de son avec les différentes techniques disponibles : additive, PSOLA, CHANT, DirectSignal et GRET.

Bien qu'il soit ici question de CHANT, il serait plus juste de parler de synthèse par fonctions d'onde formantique. En effet, aucune des règles de CHANT n'est disponible dans Diphone Studio et le contrôle des paramètres de synthèse est extrêmement réduit.

Un autre gros inconvénient de CHANT dans Diphone Studio pour la synthèse de son vocaux est que seul les résultats obtenus par une analyse par modèle de résonnance peuvent être utilisés. CHANT est donc ici plutôt orienté vers une utilisation pour la synthèse

---

136 IOVINO, Francisco ; LAURSON, Mikael, *PatchWork PW-Chant reference*, Paris : IRCAM, 1996.

137 LOIZILLON, Guillaume, *Diphone Studio: User Manual and Tutorial*, Paris : IRCAM, 1999.

de sons percussifs, le modèle par fonctions d'onde formantique permettant d'obtenir de très bons résultats pour ce cas de figure.

Si CHANT est implémenté dans Diphone Studio de nos jours, ce n'est certainement pas pour effectuer de la synthèse vocale. Il faut donc retenir de ce cas que la synthèse par fonctions d'onde formantique peut avoir d'autres utilisations que celles pour lesquelles elle a été créée à l'origine comme le souligne Xavier Rodet :

> « On aura compris que le programme CHANT n'a pas pour ambition restrictive l'étude de la voix, ou plutôt que nous ne considérons pas la voix comme un objet simple, univoque. Mais que bien au contraire la voix se présente pour nous comme un point de départ d'une richesse et d'une complexité à la fois uniques et incontournables par l'exemplarité de ses productions et leur grande diversité, notamment par la mise à jour de la notion constitutive d'articulation.[138] »

b) Chant dans Max/MSP

Max/MSP est aujourd'hui l'un des outils les plus utilisé pour le traitement du signal orienté vers la musique[139]. Une bibliothèque d'objets regroupant quelques unes des règles et des procédés de CHANT a été créée par Francisco Ionivo.

A la différence de CHANT ou de PatchWork, Max/MSP est un logiciel de traitement en temps réel. La bibliothèque d'objets de Fransisco Iovino y est donc utilisée de façon instantanée pour produire des sons de type vocaux. L'avantage d'un tel système est qu'il peut par exemple être connecté à un périphérique Midi pour une utilisation en direct par le musicien. Son gros inconvénient est qu'il ne permet pas de produire des résultats aussi précis que ceux générés par un système en temps différé où il est possible de préciser à l'avance les voyelles utilisées, les nuances ou encore le phrasé.

Afin de créer un patch reproduisant le modèle de synthèse de CHANT, il est nécessaire dans un premier temps d'utiliser cinq générateurs de FOFs et de les connecter en parallèle. Pour ceci, il est possible de créer un *patcher* récupérant tous les paramètres nécessaires à la synthèse (largeurs de bande, amplitude et fréquences des formants, etc.) et qui les envoie aux différents générateurs de FOFs :

---

138 RODET, Xavier ; BARRIERE, Jean-Baptiste ; POTARD, Yves, *Rapport de recherche n°35 : Chant, de la synthèse de la voix à la synthèse en général*, Paris : IRCAM, 1985, p. 20.

139 http://cycling74.com/products/Max/Mspjitter/ Site officiel commercial du programme Max/MSP (en ligne le 04/09/2010).

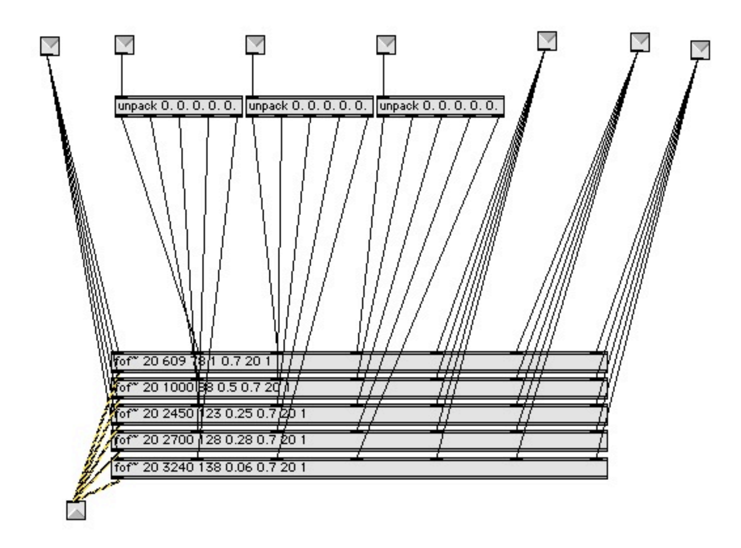


Fig. 46 – Patcher « fofs » récupérant depuis le patch principal les différents paramètres nécessaire au fonctionnement des générateurs de FOFs connectés en parallèle.

La bibliothèque de CHANT pour Max/MSP met à disposition de l'utilisateur les différentes règles suivantes :

- Calcul automatique des amplitudes des formants : *chant_autoamp*
- Correction de l'amplitudes des formants en fonction de l'effort vocal : *chant_spcor*
- Calcul automatique des largeurs de bande des formants : *chant_autobw*
- Vibrato : *chant_vibrato*
- Jitter : *chant_jitter*

En les utilisant, il n'est nécessaire de donner que les fréquences des formants pour que le système fonctionne, les autres paramètres sont calculés automatiquement. Un patch utilisant la bibliothèque CHANT de Max/MSP peut avoir la forme suivante :

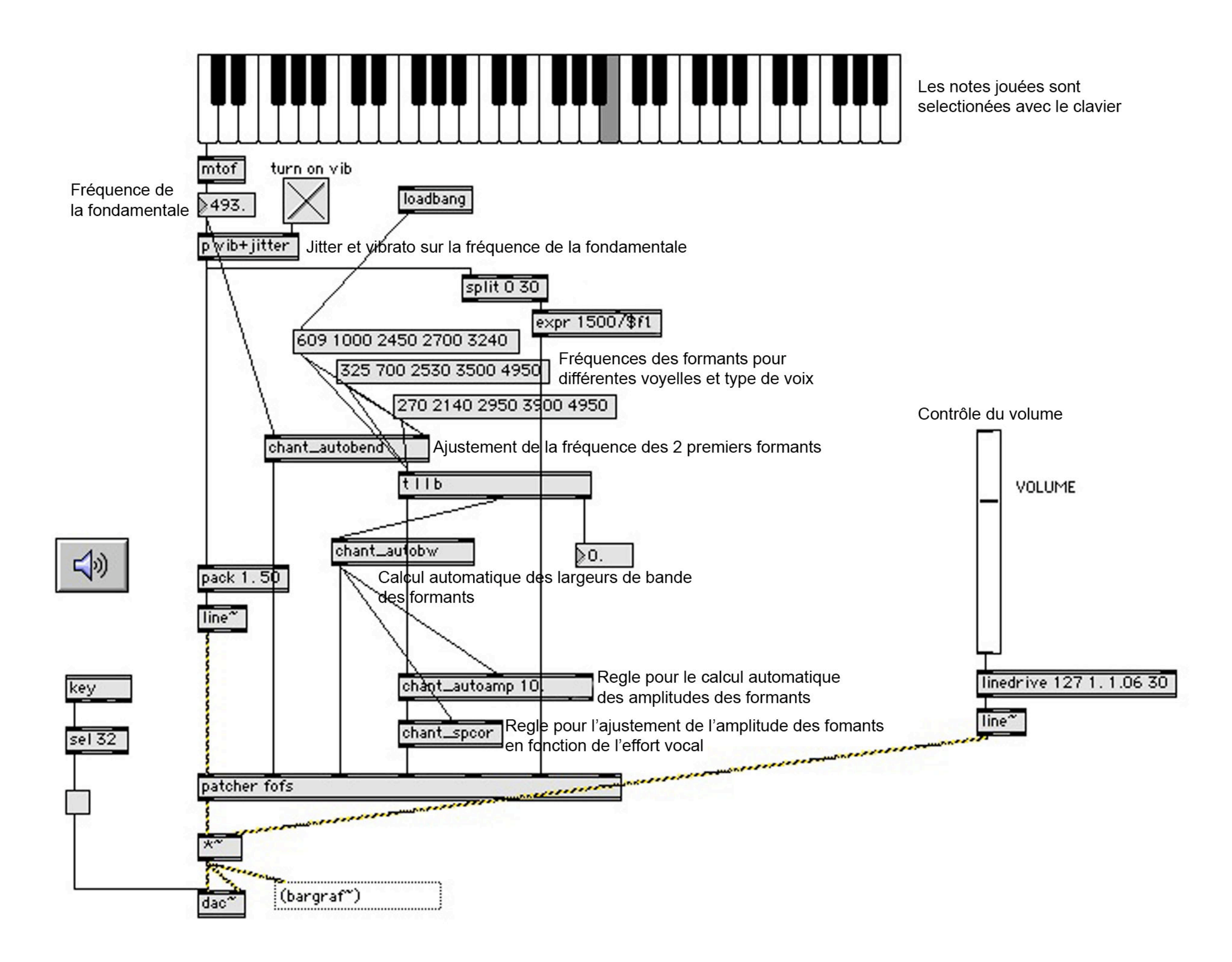
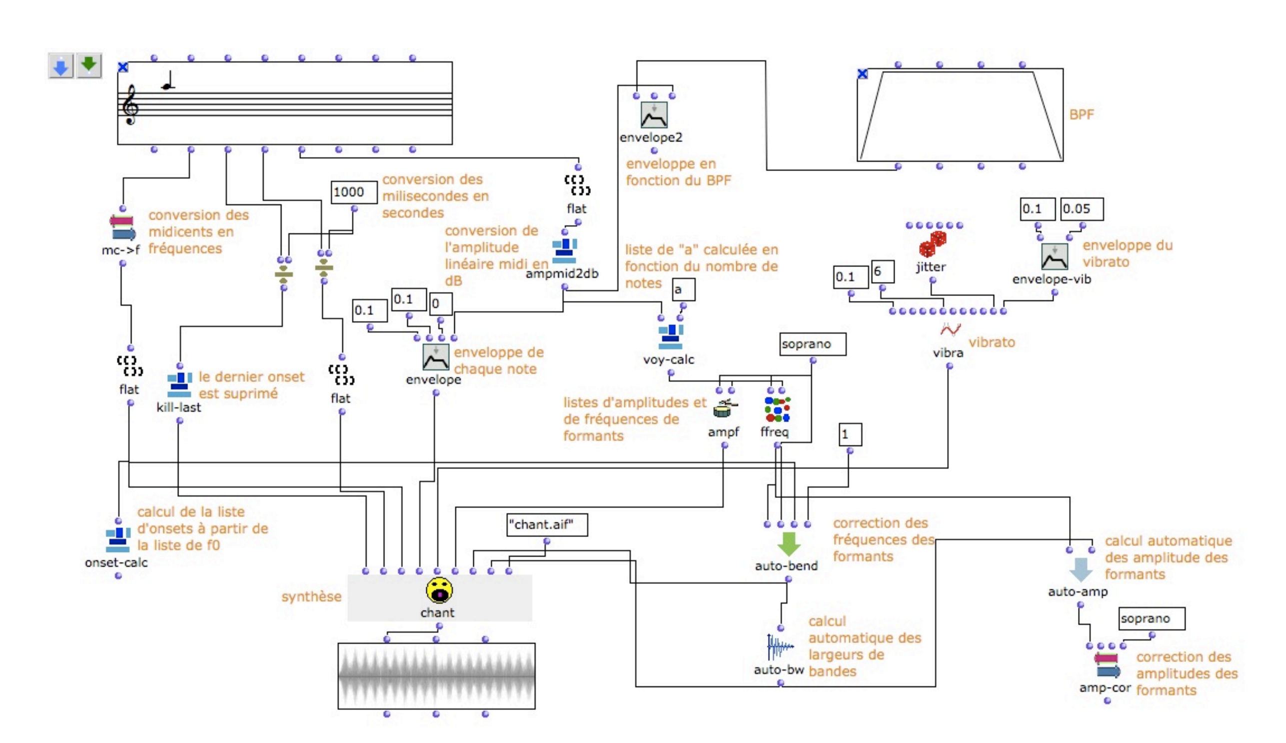


Fig. 47 – Patch utilisant la bibliothèque de CHANT pour Max/MSP.

La synthèse par fonctions d'onde formantique, et par conséquent le programme CHANT, ont été beaucoup utilisés entre le milieu des années quatre-vingt et la fin des années quatre-vingt-dix dans la production musicale contemporaine en particulier à l'IRCAM. Cet outil, à l'origine conçu pour la synthèse de la voix chantée s'est révélé être aussi particulièrement efficace dans la synthèse de sons percussifs suivis d'une résonnance (percussions, instruments à cordes princées et frappées, etc.). C'est d'ailleurs pour ce type d'utilisation qu'il est disponible dans le programme Diphone Studio avec le système d'analyse ResAn.

Dans la partie suivante, un exemple non-exhaustif d'implémentation de CHANT dans le programme OpenMusic réalisé par nos soins est présenté.

III. Implémentation possible et utilisation du programme CHANT dans OpenMusic : la bibliothèque *chant-lib*



Exemple d'utilisation de la bibliothèque *chant-lib* dans OpenMusic

La synthèse par fonctions d'onde formantique telle qu'elle est implémentée dans le programme CHANT permet d'obtenir des résultats d'une très grande qualité pour la synthèse de la voix chantée. En dépit de ce constat, il a été montré que depuis sa ré-implémentation dans PatchWork en 1992, le programme CHANT (et toutes ses fonctions) n'a jamais été porté dans son intégralité sur des plateformes actuelles.

En effet, bien que la bibliothèque de CHANT pour Max/MSP présentée dans (II)-(D)-(b) permette encore l'utilisation de certaines des règles et des fonctions de CHANT, elle est loin d'offrir toutes les possibilités du programme d'origine notamment à cause du fonctionnement en temps réel de Max/MSP.

La version de CHANT pour PatchWork était particulièrement complète et interactive. La bibliothèque de CHANT pour Max/MSP s'en est d'ailleurs très certainement inspirée. PatchWork fut progressivement remplacé par OpenMusic à partir 1998 laissant à l'abandon cette collection d'outils particulièrement efficaces pour la synthèse de la voix.

OpenMusic et PatchWork possèdent beaucoup de points communs dans leur architecture, ils sont par exemple tous deux programmés en Common LISP. Il est par conséquent envisageable d'imaginer que le portage de certains des éléments de la bibliothèque *PW-Chant* sur OpenMusic se fasse de façon assez aisée. Une telle opération permettrait de redonner accès au synthétiseur CHANT par le biais d'un logiciel *open source* et donc disponible à tous.

Dans ce chapitre, une proposition non exhaustive d'implémentation du programme CHANT dans OpenMusic réalisé par nos soins va être décrite. Les outils créés seront par la suite testés en essayant de reproduire de la façon la plus fidèle possible *L'air de la reine de la nuit* de *La flute enchantée* de Mozart synthétisé par l'équipe de Xavier Rodet à l'IRCAM en 1984.

A. Le programme OpenMusic et la synthèse par Fonctions d'Ondes Formantique

a) Fonctionnement et architecture de OpenMusic

o Le Common LISP à la base de l'architecture d'OpenMusic[140] [141]

Avant de décrire et d'expliquer le fonctionnement d'OpenMusic, il est important de redonner quelques notions sur le langage à la base de sa structure : le Common LISP. LISP est l'une des plus anciennes familles de langage de programmation impérative. Les langages LISP furent fortement utilisés dans les années soixante-dix et quatre-vingt dans les domaines de recherche sur l'intelligence artificielle. De nos jours, on les retrouve principalement dans la programmation pour la musique assistée par ordinateur et dans le domaine des finances.

OpenMusic est programmé dans un dialecte particulier des langages de la famille LISP : le Common LISP. C'est une version standardisée par ANSI[142] des variantes divergentes de LISP qui l'ont précédé.

L'une des grandes spécificités des langages LISP est le traitement de liste. Une liste est constituée d'éléments entourés par deux parenthèses et séparés par des espaces selon le modèle suivant : `(a b c d 1 2 3 4 "pim" "pam" "poum")`. Des listes peuvent en contenir d'autres et ce à l'infini : `((a b) (c d))`.

Les fonctions pour le traitement des listes sont elles-mêmes contenues dans des listes. En LISP, une fonction est invoquée en plaçant son nom au début de la liste à traiter. Par exemple pour effectuer l'opération 1+2, on écrira : `(+ 1 2)`.

Tout comme dans l'exemple donné ci-dessus, des listes contenants des fonctions peuvent être elles aussi emboîtées. Ainsi, l'expression suivante : `(- 1 (+ 1 2))` correspond à 1-(1+2).

Les commentaires sont introduits avec des points-virgules s'ils ne sont que sur une ligne et sont encadrés par `#| commentaire |#` s'ils s'étalent sur plusieurs lignes.

---

140 WINSTON, Patrick-Henry ; HORN, Berthold-Klaus-Paul, *LISP Third Edition*, Reading (Massachusetts) : Addison-Wesley, 1989.

141 http://common-lisp.net/ Site officiel de la communauté travaillant sur Common Lisp (en ligne le 04/09/2010).

142 American Standart National Institute.

Les codes LISP étant codés en ASCII[143] dans OpenMusic, il est nécessaire de normaliser le nom des fonctions et des variables utilisées ainsi que les lignes de commentaires pour qu'ils soient compatibles avec ce standard. Par conséquent, l'utilisation des accents et des caractères spéciaux tel que *ç* est prohibée dans les codes LISP.

Enfin, il est important de préciser que OpenMusic utilise le standard CLOS qui fait partie de Common LISP. CLOS étant l'abréviation *Common Lisp Object System*, on comprend qu'il est ici fait référence à de la programmation orientée objet qui est abondement utilisée dans OpenMusic.

- **Éléments fondamentaux d'OpenMusic[144] [145]**

OpenMusic est développé à l'IRCAM depuis 1998 par Gérard Assayag, Carlos Agon et Jean Bresson. C'est un environnement de développement graphique permettant la création de patchs pour la composition musicale assistée par ordinateur (CAO). De la même manière que dans Max/MSP, un patch est créé en connectant entre eux différents objets représentés par des icones. Certains bénéficient d'une interface graphique dédiée, comme les éditeurs de partitions ou encore les fichiers audio qui peuvent être représentés par leur forme d'onde, on parle alors de classes (cf. partie suivante : *Bibliothèque, package, fonctions et classes*).

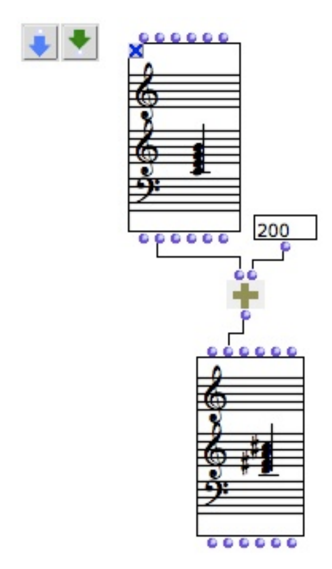


Fig. 48 – Exemple de patch effectuant une transposition dans OM – D'après le patch « exemple-patch.omp[146] ».

143 American Standart Code for Information Interchange.

144 HADDAD, Karim, *OpenMusic User's Manual*, Paris : IRCAM, 2003.

145 http://recherche.ircam.fr/equipes/repmus/OpenMusic/index.html (en ligne le 04/09/2010).

146 Disponible sur le cd à cd/om.

OpenMusic (OM) est développé sous une licence GNU/GPL[147] et est basé sur le langage Common Lisp tout comme son prédécesseur PatchWork. L'interface graphique interprétant les codes Lisp ainsi que l'utilisation du standard CLOS[148] rendent son utilisation particulièrement intuitive. Avec CLOS, la programmation dans OpenMusic est donc « orientée-objet ».

La plateforme de travail à la base d'OpenMusic est appelée « Workspace ». Elle se présente sous la forme d'un explorateur de dossiers dans lequel il est possible de trouver des patchs (damiers verts) et des dossiers contenants des patchs ou d'autres dossiers comme le montre la figure suivante :

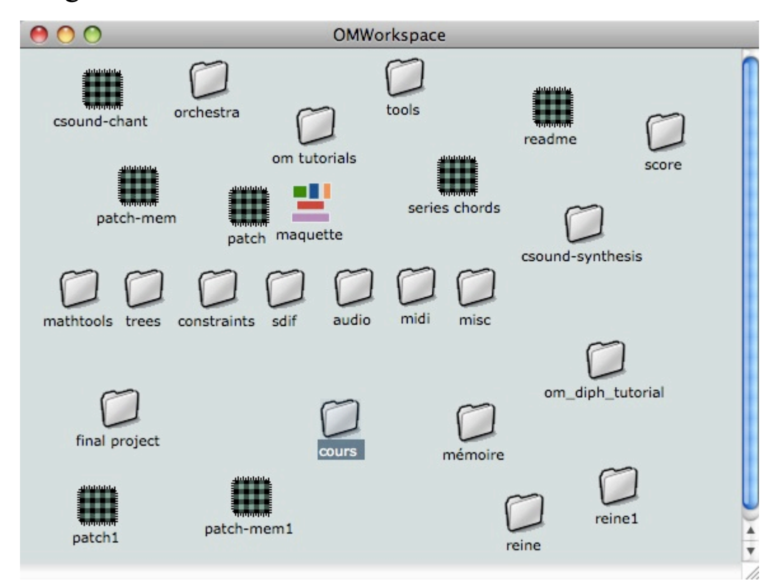


Fig. 49 – Exemple de Workspace dans OpenMusic.

Un autre élément fondamental de OM est le *listener*. C'est une interface de communication entre l'utilisateur et le noyau *Digitool Macintosh Common Lisp* avec lequel le programme fonctionne :

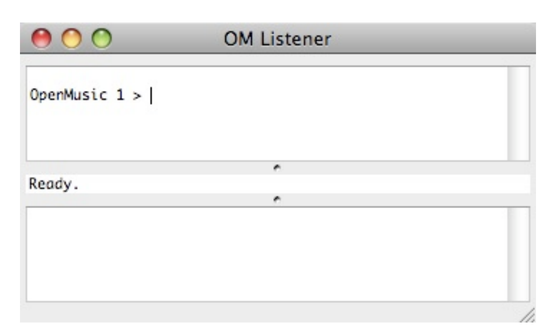


Fig. 50 – Listener dans OpenMusic.

147 Licence Publique Générale.

148 Common Lisp Objet System.

○ Bibliothèques, packages, fonctions et classes[149] [150]

Les objets utilisés dans les patchs de OM sont répertoriés dans différents packages qui peuvent être visualisées dans la fenêtre *Library* comme le montre la figure 53. Certains d'entre eux sont directement intégrés à la base d'OpenMusic tandis que d'autre peuvent être ajoutés en fonction des besoins de l'utilisateur.

On trouve sur la partie supérieure des icones représentant un objet dans OM ses différentes entrées et sur la partie inférieure ses sorties symbolisées par des boules appelées *slots* dans le manuel d'OpenMusic. Il existe deux types d'objet dans OpenMusic héritant directement du standard CLOS pour la programmation orientée objet : les fonctions et les classes.

Les fonctions sont en mesure de prendre en entrée différents types de données (chaine de caractère, nombres, etc.) et ne peuvent retourner qu'un seul paramètre. Les fonctions sont composées de méthodes répondants chacune à un type de données. Ainsi, plus une fonction pourra prendre en paramètre un nombre important de type de données, plus elle contiendra de méthodes.



Fig. 51 – Exemple de fonction dans OM.

Les classes sont des prototypes pour certains types d'objets et permettent de grouper n'importe quel ensemble de données dans une seule entité. Les classes retournent autant de paramètres qu'elles ont d'entrées. La première entrée correspond toujours au *self* de la classe représentant l'objet dans son intégralité avec tous ses paramètres. Enfin, il est important de mentionner le principe « d'hérédité » dans le fonctionnement des classes. En effet, une classe peut hériter des propriétés d'une autre et ce à l'infini.

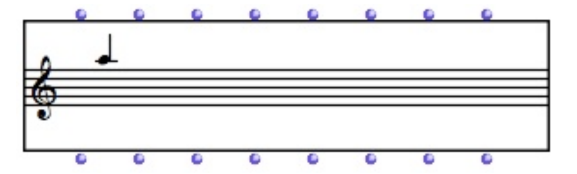


Fig. 52 – Exemple de méthode dans OM.

149 HADDAD, Karim, *OpenMusic User's Manual*, Paris : IRCAM, 2003.

150 http://recherche.ircam.fr/equipes/repmus/OpenMusic/index.html Site officiel du programme OpenMusic (en ligne le 04/09/2010).

La fenêtre *Library* d'OpenMusic classe les différents objets d'un package en fonction de son type :

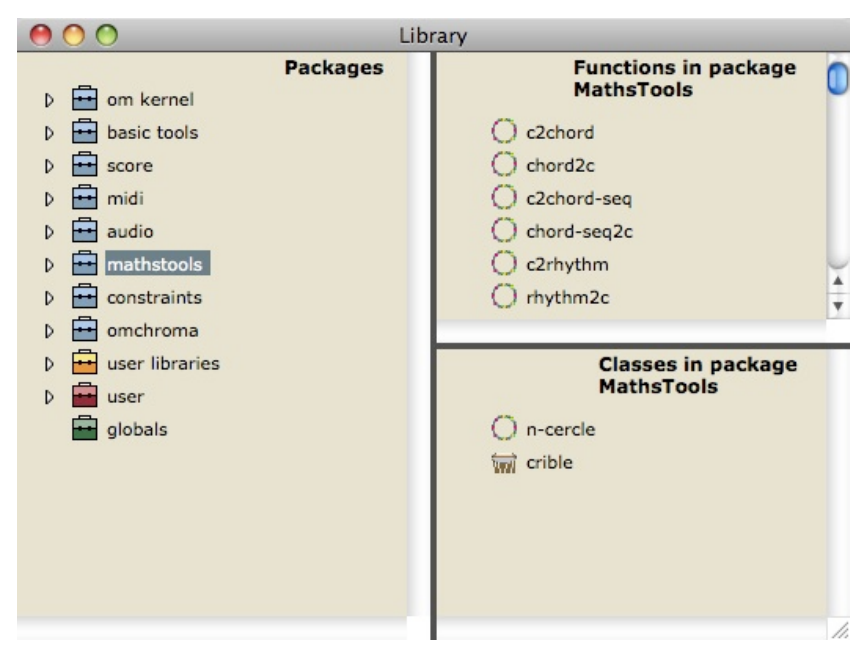


Fig. 53 – Organisation de la fenêtre *Library* dans OpenMusic.

b) Premières tentatives d'implémentation de CHANT dans OpenMusic : la bibliothèque omChroma[151] [152]

L'une des bibliothèques d'objets la plus ancienne d'OpenMusic est *omChroma*. Elle est dédiée au contrôle de système pour la synthèse sonore tel que SuperVP ou encore CSOUND. Dans sa première version de septembre 2000, elle incluait également une collection d'objets pour le contrôle de la synthèse dans CHANT. OpenMusic n'ayant pas été conçu pour effectuer des tâches de traitement numérique du signal, *omChroma* faisait appel au plugin *CHANT* contenu dans Diphone. Ce dernier devait donc être installé sur la machine exécutant l'opération décrite précédemment.

Les objets *CH-FOB-EVT* et *chant-patches* permettaient la création d'un fichier SDIF qui était envoyé et utilisé par CHANT pour générer le son de synthèse. *chant-patches*, lorsqu'il était évalué, permettait la sélection d'un modèle de connexion des différents éléments de CHANT :

---

151 ASSAYAG, Gérard ; AGON, Carlos, *OpenMusic omChroma: Paradigms for the high-level musical control of sonic processes using omChroma*, Paris : IRCAM, 2000, p. 29-30.

152 BRESSON, Jean, *Sound Processing in OpenMusic : actes de la neuvième Conference on Digital Audio Effects (DAFx-06), Montréal, 18-20/09/2006*, Paris : IRCAM, p. 4.

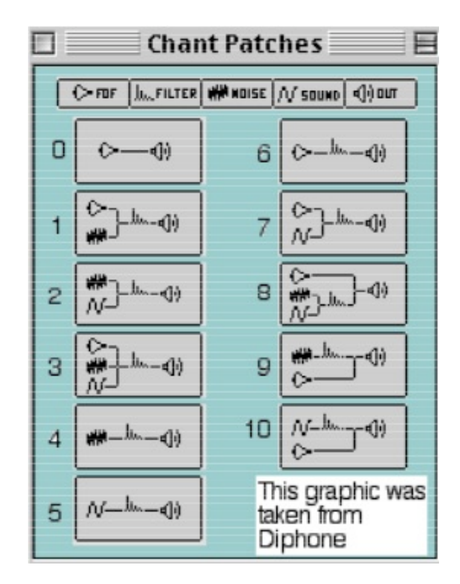


Fig. 54 – Modèles de connexion des différents éléments de CHANT dans le plugin *CHANT* de Diphone Studio[153].

L'étape de production du fichier SDIF et de synthèse était menée à bien par l'objet *synthesize* qui, à la différence des deux autres, est toujours disponible dans la bibliothèque *omChroma* d'OpenMusic.

Malheureusement, les résultats obtenus dans OpenMusic pour la synthèse avec CHANT étaient loin d'égaler ceux des versions antérieures comme cela est précisé dans le manuel d'utilisation de *omChroma* :

> « Le synthétiseur SDIF de CHANT ne permet pas d'avoir un contrôle de haut niveau sur ses paramètres. Dans le but de produire des sons plus convaincants, il est nécessaire de trouver de meilleures stratégies de contrôle, d'avoir des instruments plus élaborés ainsi que des paradigmes plus sophistiqués.[154] »

153 ASSAYAG, Gérard ; AGON, Carlos, *op. cit*, p. 30.

154 ASSAYAG, Gérard ; AGON, Carlos, *op. cit*, p. 30, citation traduite de l'anglais : « The Chant SDIF synthesizer does not provide any high-level controls for these parameters. In order to generate more convincing sounds, it is necessary to find better control strategies, have more elaborated instruments and more sophisticated paradigms. ».

B. Implémentation possible de CHANT dans OpenMusic : la bibliothèque *chant-lib*

Bien que quelques tentatives d'implémentation de CHANT dans OpenMusic aient été faites, elles n'ont jamais totalement été en mesure de remplacer les outils existants dans le passé. Il a été montré dans la partie précédente que la réutilisation du plugin *CHANT* de Diphone Studio n'est pas la meilleure solution.

Il est aisé de voir d'après les observations faites dans le paragraphe (II)-(C) que la complexité du programme CHANT réside plus dans ses différentes règles que dans son procédé de synthèse. En effet, il a été montré qu'il est particulièrement simple d'implémenter un synthétiseur par fonctions d'onde formantique dans CSOUND en connectant en parallèle plusieurs *opcodes* `fof` (cf. (II)-(A)-(c) et « fof-sop.csd[155] »).

Interfacer OpenMusic avec CSOUND est une opération simple à effectuer avec la collection d'outils disponibles dans la bibliothèque *om2csound*. Il était donc tout à fait possible d'imaginer la création d'une collection d'objets permettant l'implémentation de CHANT dans OpenMusic grâce à CSOUND.

Ces outils ont ainsi été regroupés dans une bibliothèque nommée *chant-lib* et fonctionne d'une manière semblable à ceux autrefois disponibles dans PatchWork.

Dans cette partie, nous présentons la démarche que nous avons adoptée pour implémenter CHANT dans CSOUND.

a) Description générale de la bibliothèque *chant-lib*

L'implémentation des différents éléments de CHANT dans OpenMusic passe ici par une collection de fonctions répertoriées dans la nouvelle bibliothèque que nous avons construite : *chant-lib*. Elles gravitent toutes autour d'une fonction principale appelée *chant*. Celle ci permet de récupérer l'ensemble des paramètres nécessaires à la synthèse, de les transformer en partition CSOUND et de jouer cette partition avec l'orchestre « chant.orc » dont le fonctionnement sera détaillé dans (III)-(B)-(d). La fonction *chant* compte au total neuf entrées. Ses principaux paramètres sont la liste des durées des notes, la liste des débuts des

155 Disponible sur le cd à cd/csound et dans l'annexe n°5.

attaques des notes et la liste des fréquences f_0. Les autres entrées permettent de connecter les différentes règles et fonctions calculant les valeurs des largeurs de bandes, fréquences et amplitudes des formants, vibrato, etc.

Les autres objets de la bibliothèque *chant-lib* permettent de générer les paramètres de la synthèse de la même manière que dans les versions antérieures de CHANT et en respectant un modèle semblable à celui de la bibliothèque *PW-Chant* de PatchWork :

- *jitter* : contrôle des paramètres des micros-fluctuations de la fréquence de la fondamentale (cf. (III)-(B)-(e)).
- *vibra* : contrôle des paramètres du vibrato (cf. (III)-(B)-(e)).
- *envelope* : crée une enveloppe de type attaque – maintien – chute (cf. (III)-(B)-(f)).
- envelope2 : crée une enveloppe à partir d'un BPF (cf. (III)-(B)-(f)).
- *envelope-vib* : crée une enveloppe pour l'amplitude du vibrato (cf. (III)-(B)-(e)).
- *ampf* : génère une liste d'amplitudes de formants en fonction d'une voyelle et d'un type de voix (cf. (III)-(B)-(g)).
- *ffreq* : génère une liste de fréquences de formants en fonction d'une voyelle et d'un type de voix (cf. (III)-(B)-(g)).
- *bdwth* : génère une liste de largeurs de bandes en fonction d'une voyelle et d'un type de voix (cf. (III)-(B)-(g)).
- *auto-bw* : calcul automatique des largeurs de bandes des formants (cf. (III)-(B)-(h)).
- auto-bend : ajustement automatique de la fréquence des deux premiers formants (cf. (III)-(B)-(h)).
- *auto-amp* : calcul automatique des amplitudes des formants (cf. (III)-(B)-(h)).
- *amp-cor* : correction automatique des amplitudes des formants en fonction de l'effort vocal (cf. (III)-(B)-(i)).
- *ampmid2db* : convertie une liste d'amplitude linéaire en une liste d'amplitude logarithmique en décibels (cf. (III)-(B)-(j)).
- *onset-calc* : calcul automatique de l'*onset* en fonction de la durée des notes (cf. (III)-(B)-(k)).
- *voy-calc* : génère une liste de la même voyelle égale au nombre de notes à jouer (cf. (III)-(B)-(k)).
- *kill-last* : supprime le dernier élément d'une liste (cf. (III)-(B)-(k)).

La figure 55 présente un exemple simple de synthèse utilisant quelques unes des fonctions présentées ci-dessus. Deux notes de fréquences 440 et 550 Hz sont synthétisées. La voix utilisée est une voix de soprano et la voyelle un « a » :

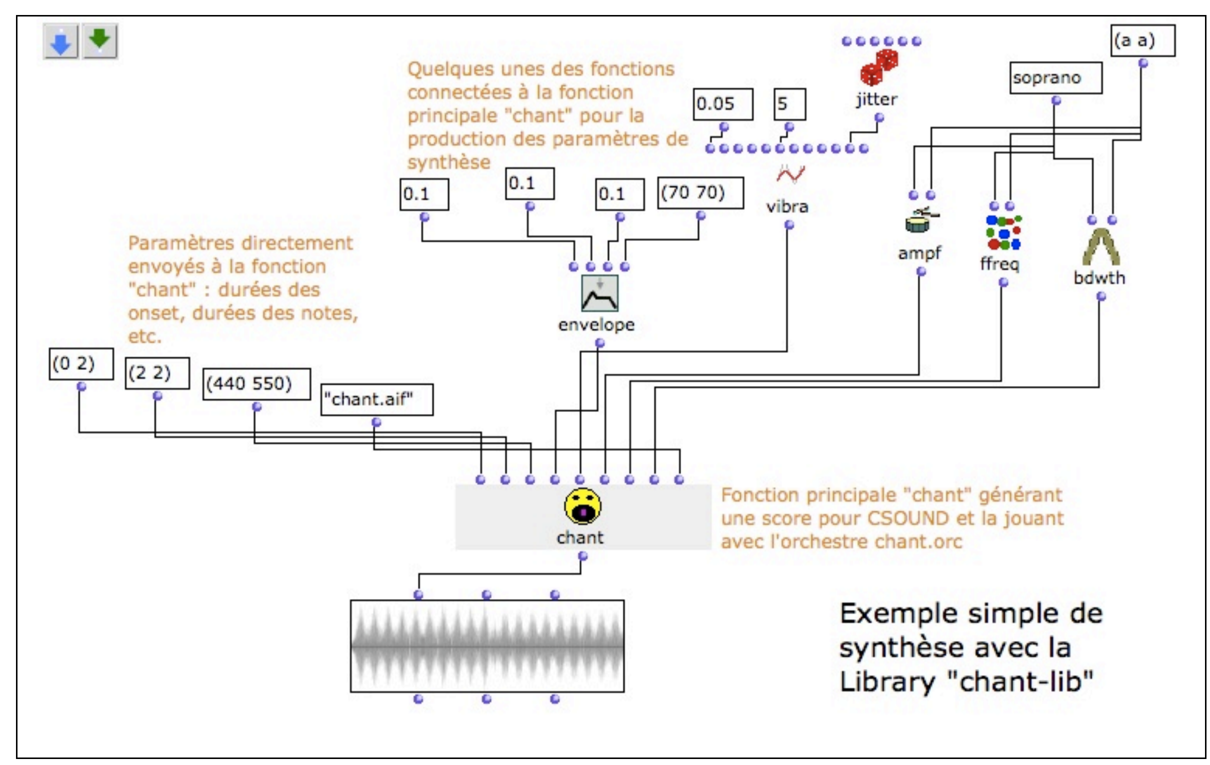


Fig. 55 – Exemple simple de synthèse avec la bibliothèque *chant-lib* en utilisant la fonction principale *chant* et quelques unes des autres fonctions disponibles. Le patch produit ici deux notes de fréquence 440 et 550 Hz chantées par une soprano sur la voyelle « a » – D'après le patch « synthese-base.omp[156] ».

L'exemple donné précédemment reste particulièrement simple. Avant d'expliquer de façon précise le fonctionnement et la construction de chaque fonction de la bibliothèque *chant-lib* et leur rôle dans la synthèse, il est important de décrire le cheminement du raisonnement qui a amené à la conception de l'architecture globale des éléments de cette dernière.

b) La bibliothèque *om2csound* : modèle de départ dans la réalisation du système

o Présentation du matériel d'origine

OpenMusic offre plusieurs possibilités d'interfaçage avec CSOUND. On peut par exemple citer les objets contenus dans la bibliothèque *chroma-classes*[157] qui permettent l'utilisation de certains *opcodes* de CSOUND dans un patch OM. Ils se présentent sous la forme de classes, rattachées à un orchestre CSOUND contenant l'*opcode* à émuler. L'étape de synthèse se fait dans CSOUND avec l'objet *synthesize* comme le montre la figure 56 :

156 Disponible sur le cd à cd/om.

157 Aucune documentation n'est actuellement disponible pour cette bibliothèque.

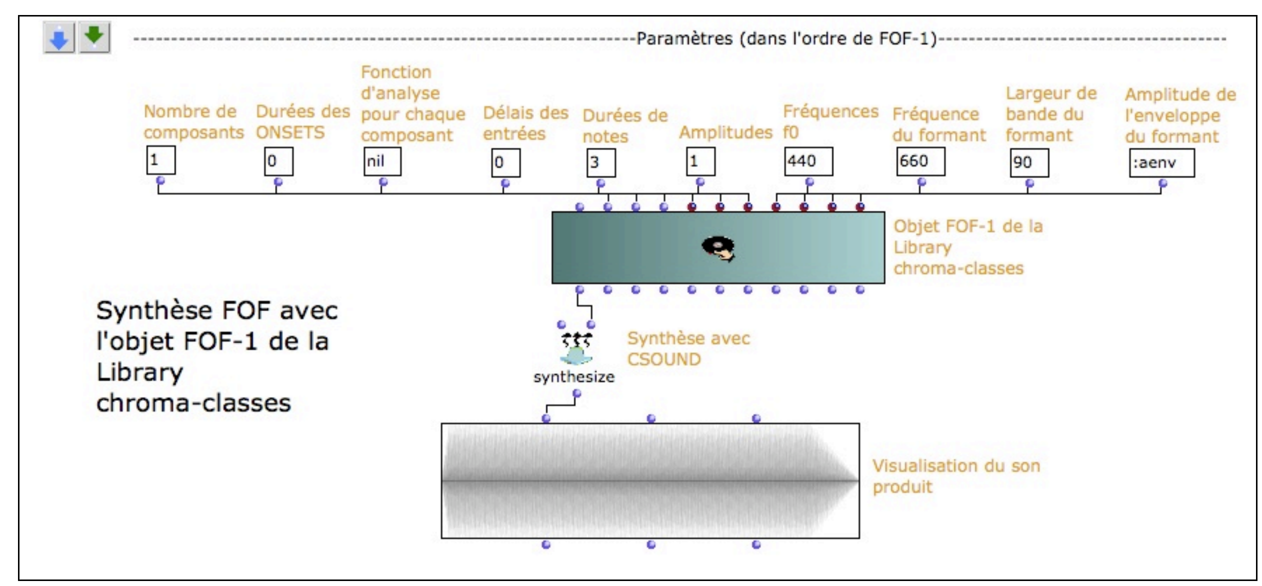


Fig. 56 – Exemple d'utilisation de l'objet *FOF-1* de la bibliothèque *chroma-classes* – D'après le patch « fof-chroma.omp[158] ».

Cette solution s'est révélée inadaptée dans notre cas dans la mesure où elle limitait fortement la liberté dans l'utilisation d'orchestres CSOUND complexes.

L'autre modèle d'interfaçage d'OpenMusic avec CSOUND est celui proposé par la bibliothèque *om2csound*[159]. Dans ce cas, OpenMusic sert de support à la création de partitions CSOUND qui sont ensuite jouées avec un orchestre donné.

La fonction *auto-editsco* de *om2csound* permet de générer une partition CSOUND. Elle prend en argument des listes contenant les différentes tables de fonctions utilisées, les notes à jouer ainsi que les différents paramètres qui leurs sont relatifs. Elle retourne le chemin d'accès de la partition créée.

Les tables de fonctions sont générées grâce à la fonction *table*. Ses quatre premiers arguments sont : le numéro de la fonction, sa durée d'exécution (zéro si inconnue), le nombre de point dans la table et le GEN utilisé. Les autres informations sont données aux dernières entrées sous la forme d'une liste.

La liste de notes et les paramètres d'un instrument sont créés avec la fonction *instrument1*. Ses arguments sont le numéro de l'instrument, une liste des onsets, une liste de la durée des notes et les différents p-fields.

Une fois la partition créée, elle peut être jouée dans CSOUND avec la fonction *csound-synth* qui prend en paramètres les chemins d'accès de la partition et de l'orchestre. Il

---

158 Disponible sur le cd à cd/om.

159 HADDAD, Karim, *om2csound : bibliothèque de modules pour la génération de scores pour Csound (version 1)*, Paris : IRCAM, 1999.

est important de préciser que la fonction *auto-editsco* peut donc être directement connectée à *csound-synth*. Il est enfin possible d'écouter et de visualiser le son produit avec la classe *sound*.

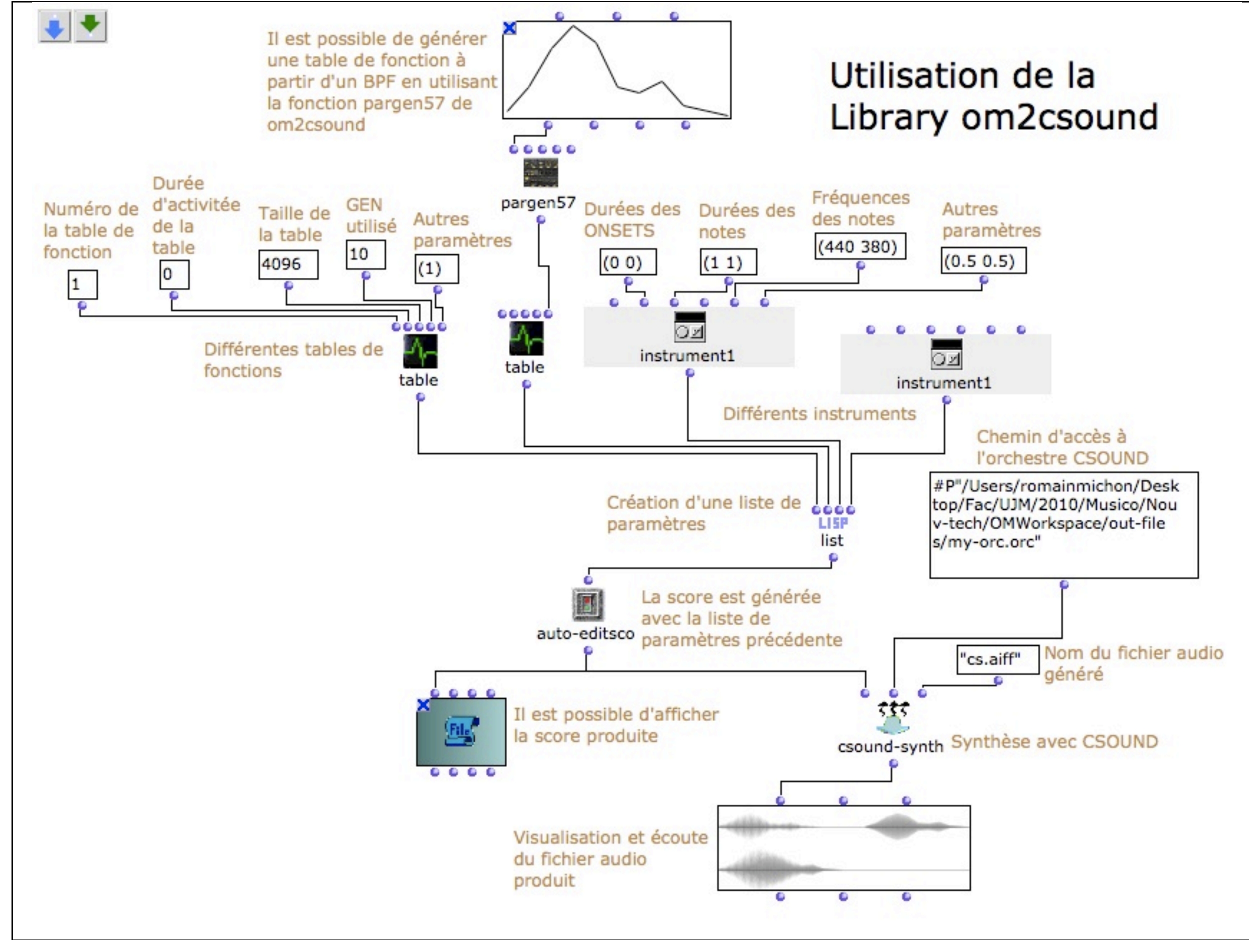


Fig. 57 – Exemple d'utilisation de la bibliothèque *om2csound* – D'après le patch « om2csound.omp[160] ».

La bibliothèque *om2csound* fait partie intégrante de l'architecture de la bibliothèque *chant-lib* dont il est question dans ce chapitre. Ainsi, un nombre important des fonctions de *om2csound* sont utilisées dans les différents objets de *chant-lib*. Il est primordial pour cette raison d'activer et de charger *om2csound* à chaque fois que des éléments de *chant-lib* sont utilisés dans OpenMusic.

○ Fonction pour la conversion de listes de paramètres en partition CSOUND : *exec-csound*

Les différents paramètres envoyés à la fonction *chant* sont transformés en une partition CSOUND qui est ensuite lue avec l'orchestre « chant.orc ». Cette opération est

160 Disponible sur le cd à cd/om.

effectuée par une fonction interne à la bibliothèque *chant-lib* qui n'est pas visible dans le package de cette dernière : *exec-csound*. Celle-ci s'inspire fortement des fonctions *auto-editsco* et *csound-synth* de la bibliothèque *om2csound*.

Afin de comprendre au mieux son fonctionnement, il est nécessaire d'étudier son code LISP. Celui-ci est présenté dans l'annexe n°7.

c) La fonction *chant*

La fonction *chant* permet de créer la liste de paramètres qui sera envoyée à la fonction *exec-csound* et qui sera convertie en partition. Pour cela, elle récupère et analyse les résultats des autres fonctions de la bibliothèque *chant-lib* qui sont envoyés sous formes de listes.

chant compte au total huit entrées obligatoires et une optionnelle. Chacune d'entre elles permet de faire passer un type de paramètre particulier nécessaire à la synthèse.

○ Entrées pour l'onset, la durée et la fréquence f_0

L'entrée n°1 (*onset*) permet de définir la liste de durées correspondant au minutage du début de chaque note. Cette liste est de la forme : $(t_0\ t_1\ t_2\ (t_{n-1}+t_n))$ où t est la durée totale d'un événement dans CSOUND. Par exemple, si on souhaite jouer deux notes successives ayant chacune une durée de deux secondes, *onset* aura pour valeur `(0 2)`. Si on ajoute un silence entre ces deux notes d'une durée d'une seconde, *onset* = `(0 3)`.

Dans la mesure où cette entrée n'accepte que des listes, il est nécessaire de mettre des parenthèses même lorsque une seule durée est fournie. Pour une note de quatre secondes on notera donc : `(4)`.

L'entrée n°2 (*duree*) permet de donner la liste des durées de chaque note. Si on reprend le premier exemple donné précédemment (deux notes consécutives de deux secondes), *duree* sera égal à `(2 2)`. Tout comme le paramètre *onset*, *duree* n'accepte que des listes.

L'entrée n°3 (*f0*) permet de fournir la liste des fréquences de la fondamentale en Hertz. Par exemple deux notes successives, un La3 et un Do#4 se noteront `(440 550)`. Un Mi4 (660).

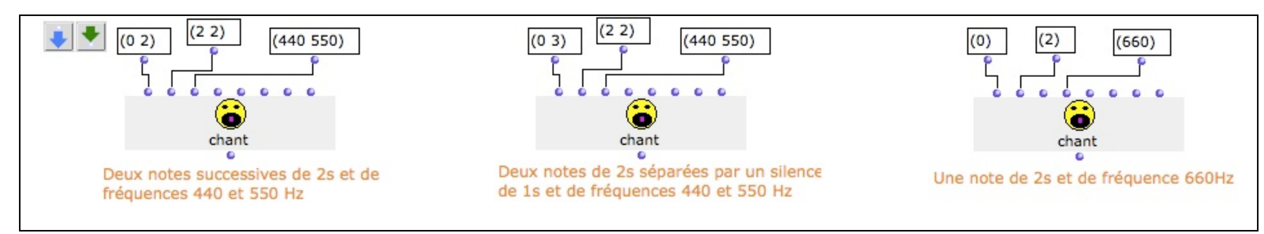


Fig. 58 – Exemples de phrases musicales avec l'objet *chant* - D'après le patch « ons-dur-freq.omp[161] ».

○ Entrées pour les paramètres de l'enveloppe, du vibrato et des listes d'amplitudes des formants

L'entrée n°4 (*envelope*) permet de connecter l'une des deux fonctions pour le contrôle de l'enveloppe des notes ou de la phrase musicale jouée : *envelope* et *envelope2* dont le fonctionnement sera décrit plus en détail dans (III)-(B)-(f). Ces fonctions permettent également d'indiquer l'amplitude de chaque note en décibels.

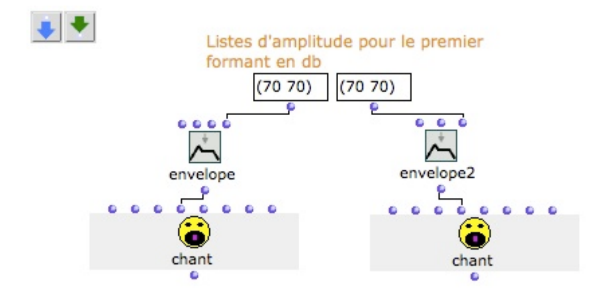


Fig. 59 – Exemples de connexions des fonctions *envelope* et *envelope2* à la fonction *chant* - D'après le patch « envelope.omp[162] ».

L'entrée n°5 (*vibra*) permet de connecter la fonction *vibra* qui contrôle les paramètres des différents types de modulation de la fréquence f_0 de la fondamentale. Il est important de préciser que la fonction *jitter* doit être connectée à *vibra* pour avoir un effet sur le son produit comme cela sera expliqué plus en détail dans (III)-(B)-(e).

L'entrée n°6 (*ampf*) permet de définir la liste de listes des amplitudes des formants du son produit. Les amplitudes données sont en décibels et doivent avoir des valeurs négatives. Elles seront ainsi soustraites à l'amplitude positive du premier formant dont les valeurs sont définies par la fonction *envelope*. L'orchestre CSOUND « chant.orc » de la bibliothèque *chant-lib* fonctionnant avec cinq générateurs de formants, il est nécessaire de fournir des listes de (5 – 1) amplitudes (l'amplitude du premier formant est donnée dans la fonction *envelope*).

---

161 Disponible sur le cd à cd/om.

162 Disponible sur le cd à cd/om.

Si deux voyelles *a* sont chantées successivement, la liste de listes d'amplitudes sera de la forme : `((-6 -32 -20 -50) (-6 -32 -20 -50))`. Dans le cas d'un seul *a* : `((-6 -32 -20 -50))`.

Enfin, en plus d'être entrée manuellement, la liste de listes d'amplitude peut-être générée avec les fonctions *ampf* et *auto-amp*.

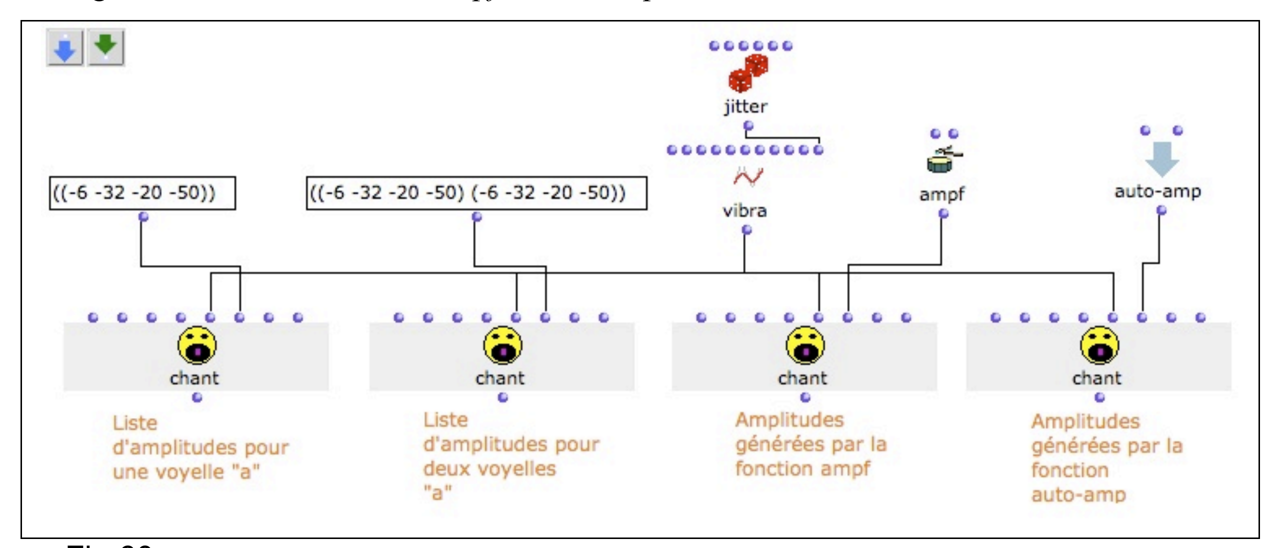


Fig.60 – Exemple d'utilisation de liste de listes d'amplitudes de formants avec modulations de la fréquence de la fondamentale - D'après le patch « ampf+vib.omp[163] ».

- **Entrées pour les listes de fréquences et de largeurs de bandes de formants**

Les entrées sept et huit permettent de définir respectivement la liste des listes des fréquences (*ffreq*) et la liste des listes des largeurs de bandes (*bdwth*) des formants. Elles fonctionnent de la même manière que l'entrée n°6. La seule différence réside dans le nombre d'éléments de chaque liste : cinq au lieu de quatre pour l'amplitude des formants.

La fonction *ffreq*, permet de donner une liste de listes de formants en fonction d'un type de voix et d'une voyelle donnée. *bdwth* effectue la même tâche pour les largeurs de bandes des formants. Enfin, la fonction *auto-bw* permet de calculer automatiquement les largeurs de bandes des formants en fonction de leur fréquence.

163 Disponible sur le cd à cd/om.

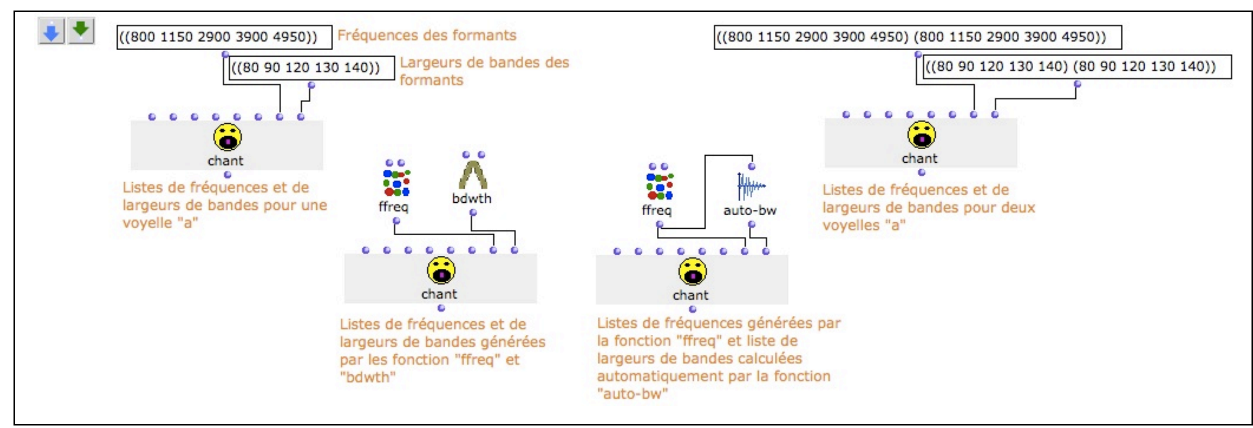


Fig. 61 – Exemples d'utilisations de liste de listes de fréquences et de largeurs de bandes de formants – D'après le patch « ffreq+bdwth.omp ».

Enfin, il existe une neuvième et dernière entrée (*out-name*). Dans la mesure où celle-ci est de moindre importance, elle est optionnelle ce qui laisse le choix à l'utilisateur de l'afficher ou pas. Elle permet de modifier le nom par défaut `("chant.aif")` du fichier audio issu du processus de synthèse. Dans la mesure où ce nom est une chaine de caractères, il doit être entouré par des guillemets.

Le fonctionnement de la fonction *chant* est présenté dans l'annexe n°8.

d) Description générale l'orchestre CSOUND « chant.orc »

CSOUND permet d'effectuer l'étape de synthèse dans notre système de CHANT pour OpenMusic. La partition « my-sco.sco » produite par la fonction *chant* est jouée dans CSOUND avec l'orchestre « chant.orc ».

Il serait inutile de présenter maintenant le code complet de « chant.orc » dans la mesure où il ne peut être compris sans avoir étudié le fonctionnement et le rôle des différentes fonctions de la bibliothèque *chant-lib*.

Néanmoins, il intéressant à ce stade de connaître le fonctionnement global de « chant.orc »[164] :

i. Les paramètres audio-numériques de l'orchestre sont déclarés. Le son produit sera en mono et aura un taux d'échantillonnage de 44100 Hz. Chaque période *k-rate* contient 4410 échantillons et il y a 10 périodes *k-rate* par seconde.

```
sr = 44100
```

164 Le code complet de « chant.orc » est disponible dans l'annexe n°9.

```
kr = 4410
ksmps = 10
nchnls = 1
```

ii. Les variables globales sont déclarées. Elles permettent principalement le maintien de certaines valeurs d'une note à une autre. Elles sont par exemples très utiles pour la création d'enveloppes globales sur toute une phrase (cf. (III)-(B)-(f)) sur la fonction *enveloppe*).

```
gilast init 0
gifre1 init 0
gifre2 init 0
[…]
```

iii. Le seul instrument de l'orchestre est initialisé. Les paramètres des cinq générateurs de FOFs sont déclarés tout comme certaines variables dont l'utilité sera expliquée ultérieurement.

```
koct = 0
kris = .003
kdur = .02
kdec = .007
iolaps = 100000
ifna = 1
ifnb = 2
itotdur = p3
iphs = 0
ifmode = 1

iinterp = p24
kenvar1 = 1
if0 = p4
aengar1 = 1
```

Comme nous l'avons expliqué dans (II)-(A)-(c), les générateurs de FOFs sont créés dans CSOUND avec l'*opcode* `fof` qui est de la forme[165] :

```
ares fof xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \
         ifna, ifnb, itotdur [, iphs] [, ifmode] [, iskip]
```

où :

- *xamp* est l'amplitude du flux de FOFs en dBfs.
- *xfund* est la fréquence f_0 de la fondamentale.
- *xform* est la fréquence du formant.
- *koct* est un index d'octaviation, ici 0 pour empêcher l'octaviation.
- *kband* est la largeur de bande du formant à -6dB en Hertz.
- *kris*, *kdur* et *kdec* sont les paramètres de l'enveloppe de chaque impulsions, ici respectivement 0.003, 0.02 et 0.007.

165 VERCOE, Barry, « FOF », *The canonical CSOUND Reference Manual, Version 5.09, éd. sous la direction de Barry Vercoe*, Cambridge : MIT, 2005, p. 750-751, disponible en ligne : http://www.csound.com/manual/ (en ligne le 04/09/2010).

- *iolaps* est la taille de l'espace réservé dans la mémoire lors du chevauchement de données, ici 100000.
- *ifna* et *ifnb* sont les numéros des tables de fonctions utilisées pour le calcul des impulsions et de leur enveloppe. *ifna* est une sinusoïde (GEN10) et *ifnb* une sigmoïde (GEN19) (cf. (II)-(A)-(c)).
- *itotdur* est la durée totale pendant laquelle l'*opcode* `fof` restera actif. Ici, *itotdur* est égal à la durée de la note jouée.
- *iphs* est la phase initiale de la fondamentale, ici 0.
- *ifmode* indique à l'*opcode* `fof` si la fréquence du formant sera statique ou dynamique. Sa valeur par défaut au début de l'orchestre est 1 (fréquence dynamique).
- *iskip* indique à l'*opcode* `fof` si la phase d'initialisation de chaque note est effectuée ou non. Dans le cas d'une phrase musicale où les notes sont jouées legato, *iskip* doit être égal à 1 (la phase d'initialisation est sautée), dans le cas inverse, *iskip* doit être égal à 0.

iv. Les valeurs des fréquences, des amplitudes et des largeurs de bande des formants sont attribuées à des variables afin de faciliter la lecture des procédés de traitement de ces informations. (cf. (III)-(B)-(g)).

v. Le vibrato est calculé à partir des paramètres donnés par la fonction *vibra*. (cf. (III)-(B)-(e)).

vi. Le jitter est calculé à partir des paramètres donnés par la fonction *jitter*. (cf. (III)-(B)-(e)).

vii. Si l'interpolation linéaire des formants est activée, leurs fréquences, amplitudes et largeurs de bandes sont interpolées d'une note à une autre. L'enveloppe d'amplitude décrite soit par la fonction *envelope* soit par la fonction *envelope2* est appliquée sur chaque note si l'interpolation linéaire des formants est désactivée ou sur la phrase musicale entière si elle est activée.

viii. Les amplitudes des formants sont ajustées selon l'algorithme décrit dans (II)-(C)-(f) si la fonction *amp-cor* est connectée à l'objet *chant* dans OpenMusic. En effet, il est important de rappeler que *amp-cor* est la seule fonction exécutée directement dans CSOUND dans la mesure où l'amplitude des formants varie au cours du temps en fonction de l'enveloppe.

ix. La synthèse FOF est exécutée en additionnant les signaux de cinq générateurs de FOFs.

```
ar1 fof
iamp1*kenv*aenvg*kenvar1*aengar1,aint+(if0*amod*kenv_vib)+kfond
_var, afre1i, koct, kwid1i, kris, kdur, kdec, iolaps, ifna,
ifnb, itotdur, iphs, ifmode, iskip

ar2 fof aamp2i*kenv*aenvg, aint+(if0*amod*kenv_vib)+kfond_var,
afre2i, koct, kwid2i, kris, kdur, kdec, iolaps, ifna, ifnb,
itotdur, iphs, ifmode, iskip

ar3 fof aamp3i*kenv*aenvg, aint+(if0*amod*kenv_vib)+kfond_var,
afre3i, koct, kwid3i, kris, kdur, kdec, iolaps, ifna, ifnb,
itotdur, iphs, ifmode, iskip

ar4 fof aamp4i*kenv*aenvg, aint+(if0*amod*kenv_vib)+kfond_var,
afre4i, koct, kwid4i, kris, kdur, kdec, iolaps, ifna, ifnb,
itotdur, iphs, ifmode, iskip

ar5 fof aamp5i*kenv*aenvg, aint+(if0*amod*kenv_vib)+kfond_var,
afre5i, koct, kwid5i, kris, kdur, kdec, iolaps, ifna, ifnb,
itotdur, iphs, ifmode, iskip

asort = ar1 + ar2 + ar3 + ar4 + ar5

out asort
endin
```

Les parties en orange du code correspondent aux informations relatives à l'amplitude des FOFs, les parties en mauve à la fréquence f_0 de la fondamentale. Leur signification sera donnée ultérieurement.

e) Modulations de la fréquence de la fondamentale

La fonction *jitter*

La fonction *jitter* permet de contrôler les paramètres des trois jitters connectés en parallèle (cf. (II)-(C)-(d)) ajoutant des micro-fluctuations à la fréquence de la fondamentale. Ses paramètres sont dans l'ordre :

- Les amplitudes des trois jitters exprimées comme un ratio de f_0 : *jitt1*, *jitt2* et *jitt3* dont leurs valeurs par défaut sont respectivement : 0.01, 0.02 et 0.03.
- Les fréquences de production des valeurs aléatoires des trois jitter : *jittf1*, *jittf2* et *jittf3* dont les valeurs par défaut sont respectivement : 50, 111 et 1218 Hz.

Dans OpenMusic, la fonction *jitter* se contente de rassembler les paramètres de trois jitters et de les insérer dans une liste qui pourra être envoyée à la fonction *chant* via la fonction *vibra* (cf. figure 63). En d'autres thermes, le corps de la fonction *jitter* se résume à :

```
(list jitt1 jitt2 jitt3 jittf1 jittf2 jittf3)
```

La création du jitter dans CSOUND est décrite dans l'annexe n°12.

o **La fonction *vibra***

La fonction *vibra* permet de contrôler les paramètres de la modulation périodique de la fréquence f_0 de la fondamentale. Le vibrato utilisé est du même type que celui de CHANT qui a été décrit dans (II)-(C)-(a). En effet, il est couplé avec un ensemble de jitters qui ont pour but d'ajouter des micros-fluctuations à son amplitude et à sa fréquence afin de reproduire les caractéristiques naturelles d'un vibrato vocal chez l'homme.

- *vibamp* : amplitude du vibrato exprimée comme un ratio de la fréquence de la fondamentale. Si *vibamp*=0, aucun vibrato n'est appliqué, si *vibamp*=1, la fréquence de la fondamentale oscillera entre -50% et +50% de sa valeur initiale. Des valeurs cohérentes pour *vibamp* peuvent être comprise entre 0 et 0.07. Sa valeur par défaut est 0.05.
- *vala1 et vala2* : amplitudes des deux jitters variant de façon aléatoire l'amplitude du vibrato exprimée comme un ratio de *vibamp*. Leur valeur par défaut est 0.01.
- *tvala1* et *tvala2* : fréquence de production des valeurs aléatoires modulant *vibamp*. Leur valeur par défaut est 5000Hz.
- *vibfreq* : fréquence de vibrato. Des valeurs cohérentes sont comprises entre 5 et 8 Hz. La valeur par défaut de *vibfreq* est 5Hz.
- *valf1* et *valf2* : amplitudes des deux jitters variant de façon aléatoire la fréquence du vibrato exprimée comme un ratio de *vibfreq*. Leur valeur par défaut est 0.01.
- *tvalf1* et *tvalf2* : fréquence de production des valeurs aléatoires modulant *vibfreq*. Leur valeur par défaut est 1000Hz.

Enfin, il est important de préciser que la fonction *vibra* possède deux entrées optionnelles permettant la connexion des fonctions *jitter* et *envelope-vib* comme le montre la figure 63.

Le traitement des informations par la fonction *vibra* dans OpenMusic est expliqué dans l'annexe n°10.

La création du vibrato dans CSOUND est décrite dans l'annexe n°11.

o **La fonction *envelope-vib***

Dans la réalité, l'amplitude du vibrato d'un chanteur est loin d'être constante. En effet, celle-ci est en générale beaucoup plus faible au début et à la fin d'une note qu'au

milieu. Il est donc possible de parler d'enveloppe d'amplitude de vibrato comme le montre le sonagramme suivant :

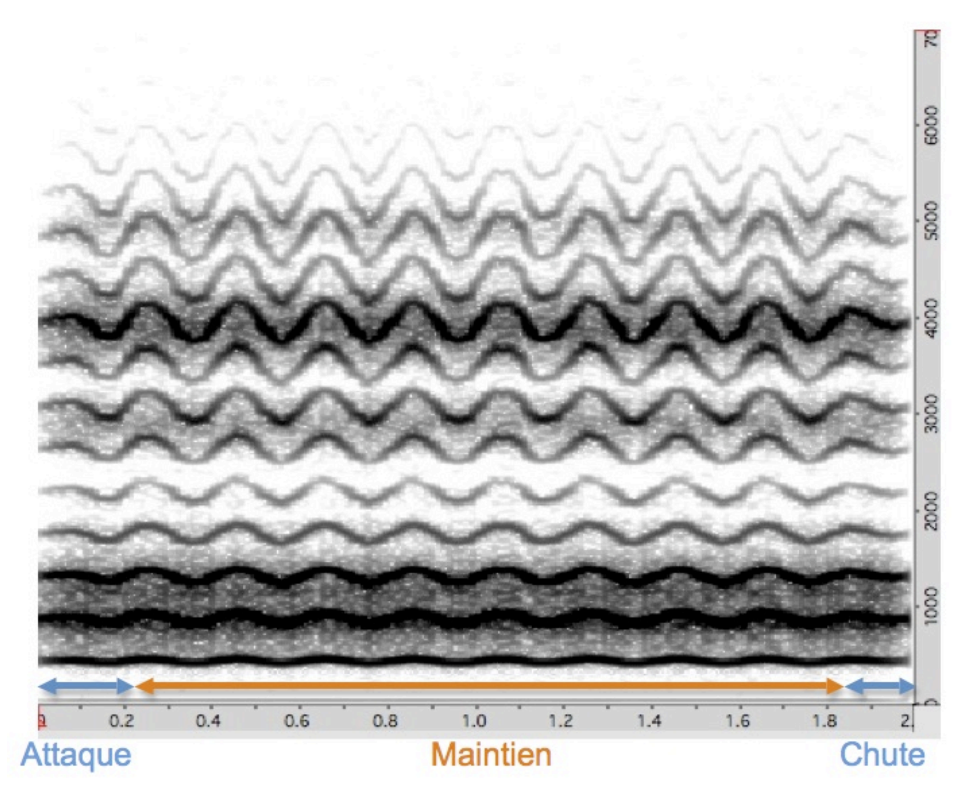


Fig. 62 – Sonagramme montrant l'enveloppe de l'amplitude du vibrato d'une note chantée.

La fonction *envelope-vib*, lorsqu'elle est connectée à la fonction *vibra* sur sa douzième entrée permet d'introduire une enveloppe de type attaque – maintien – chute à l'amplitude du vibrato. Ses deux paramètres sont donc la durée de l'attaque (*att*) et la durée de la chute (*rel*) exprimées en secondes[166]. Leur valeur par défaut est 0.1 (cf. figure 63).

Dans OpenMusic, *envelope-vib* se contente de lister les deux paramètres donnés :

```
(list att rel)
```

166 La création du vibrato dans CSOUND est décrite dans l'annexe n°11.

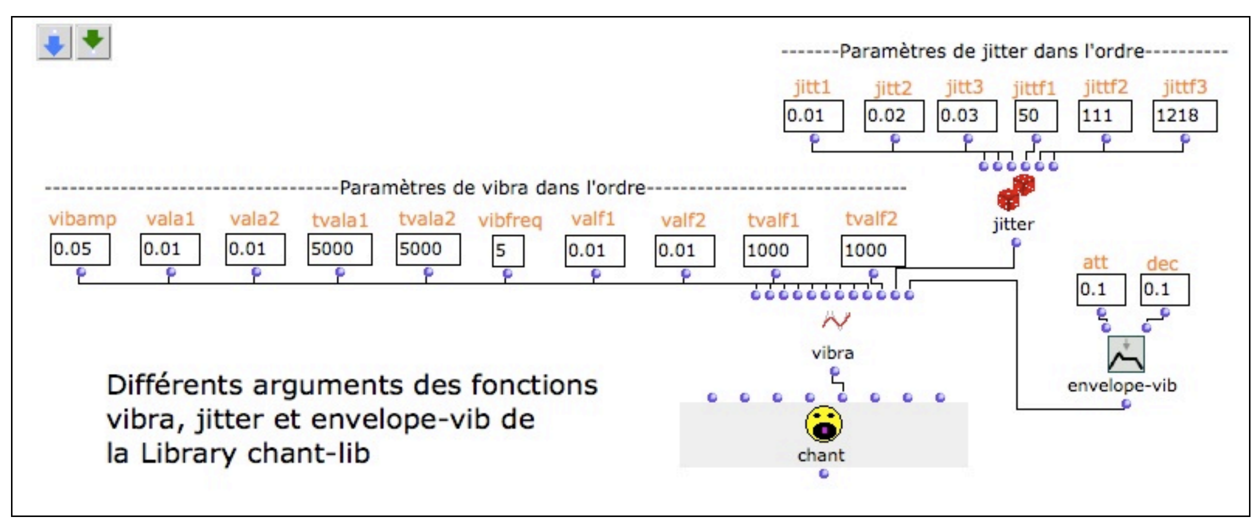


Fig. 63 – Mode de connexion et différents arguments des fonctions *vibra*, *jitter* et *envelope-vib* - D'après le patch « vibra+jitter+envelope-vib.omp[167] ».

f) Contrôle de l'enveloppe d'amplitude des notes et des phrases musicales et de l'interpolation linéaire d'une note à une autre

Plusieurs objets permettent de contrôler l'enveloppe d'amplitude des notes et des phrases musicales dans la bibliothèque *chant-lib* pour OpenMusic. Leur utilisation et leur fonctionnement vont être présentés ici.

○ La fonction *envelope*

La fonction *envelope* de la bibliothèque *chant-lib* permet d'appliquer une enveloppe d'amplitude de type attaque – maintien – chute sur les différentes notes d'une phrase musicale produite par la fonction *chant* ou sur la phrase musicale dans sa totalité. *envelope* est la seule fonction obligatoire avec *chant* de la bibliothèque *chant-lib*. En effet, elle doit être connectée à l'entrée *envelope* de la fonction *chant* pour que la synthèse ait lieu. Elle permet aussi de contrôler les paramètres de l'interpolation linéaire entre chaque note. Enfin, c'est par cette fonction que la liste des amplitudes est transmise à la fonction *chant* comme le montre la figure 64. Les paramètres de *envelope* sont donc (dans l'ordre des entrées) :

167 Disponible sur le cd à cd/om.

- *att* : La durée de l'attaque en secondes. Sa valeur par défaut est 0,1.
- *dec* : La durée de la chute en secondes. Sa valeur par défaut est 0,1.
- *tpe* : La durée de l'interpolation linéaire entre chaque note. Si celle-ci est égale à zéro, l'étape d'interpolation est sautée dans CSOUND et l'enveloppe s'applique à toutes les notes de la phrase musicale (cf. annexe n°16 sur l'interpolation linéaire de la fondamentale f_0, des amplitudes, des fréquences et des largeurs de bandes dans CSOUND). Si celle-ci est supérieure à zéro, l'enveloppe est appliquée à la phrase musicale dans sa totalité dans la mesure où les notes sont interpolées. Sa valeur par défaut est 0,15.
- *amps* : La liste des amplitudes de chaque note à jouer est en décibels. Sa valeur par défaut est `(70)`.

La tâche effectuée par la fonction *envelope* est particulièrement simple puisqu'elle consiste simplement en la création d'une liste avec les différents paramètres qui lui sont fournis :

```
(list att rel interp amps)
```

Par exemple, une liste issue de la fonction *envelope* pourrait être : `(0.1 0.1 0 (70 70))` ce qui correspond donc à deux notes d'amplitudes soixante-dix décibels, sans interpolation, avec une attaque et une chute de 0,1 secondes.

Le traitement des informations retournées par *envelope* par la fonction *chant* est décrit dans l'annexe n°14.

o La fonction *envelope2*

La fonction *envelope2* de la bibliothèque *chant-lib* permet d'appliquer une enveloppe d'amplitude à chacune des notes synthétisées par l'objet *chant*. La forme de cette enveloppe est définie par une BPF dont la sortie *self* peut-être connectée sur la première entrée de la fonction *envelope2* (*bpf0*). La forme décrite dans la BPF est alors transformée en une table de fonction CSOUND (GEN07) dont la précision (nombre de points) est définie par la valeur donnée à la troisième entrée (*pnts*) de la fonction *envelope2* comme le montre la figure 64. Sa valeur par défaut est 2048. Enfin, la deuxième entrée (*amps*) permet de transmettre la liste des amplitudes des notes à synthétiser à l'objet *chant*.

Il faut préciser qu'à la différence de la fonction *envelope*, il n'est pas possible d'utiliser l'interpolation (cf. annexe n°16 sur l'interpolation linéaire de la fondamentale f_0, des amplitudes, des fréquences et des largeurs de bandes dans CSOUND). Dans la mesure où l'amplitude de chaque note est définie au niveau de la liste des amplitudes, il serait inutile

d'appliquer une enveloppe complexe globale à l'ensemble de la phrase musicale dans le cas présent.

Le mode de fonctionnement de *envelope2* est décrit dans l'annexe n°13.

Le traitement des informations retournées par *envelope2* par la fonction *chant* est décrit dans l'annexe n°14.

La création d'enveloppes d'amplitude dans l'orchestre CSOUND « chant.orc » est expliquée dans l'annexe n°15.

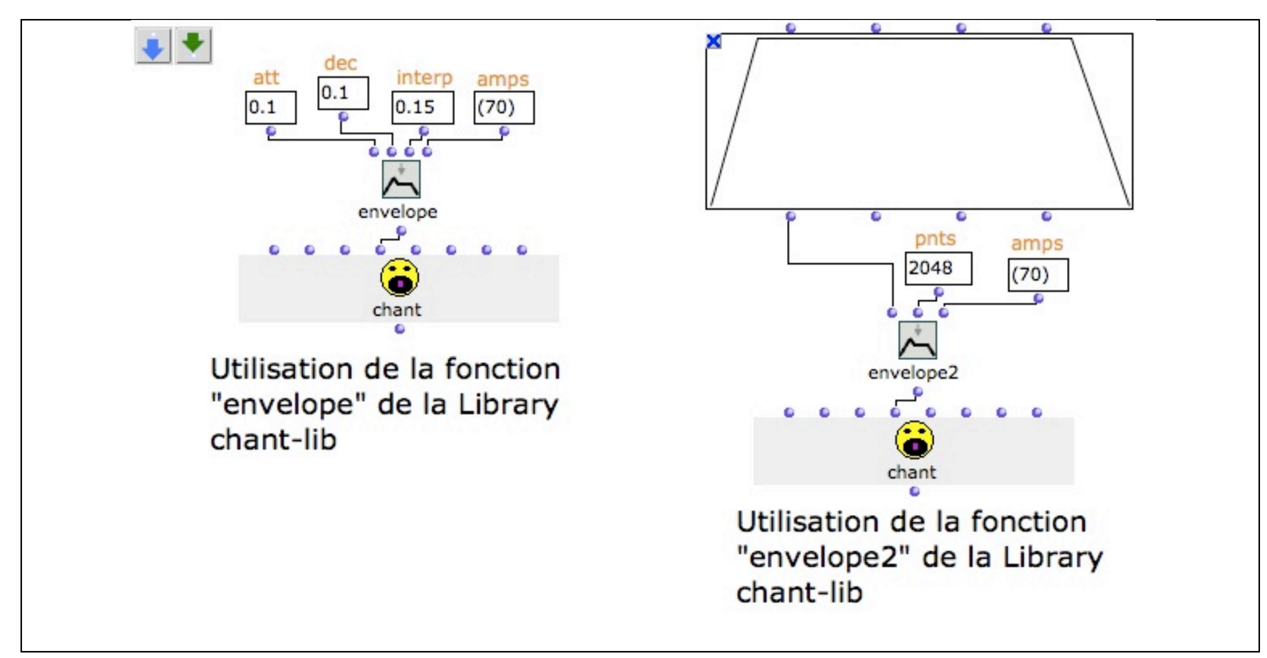


Fig. 64 – Utilisation des fonctions *envelope* et *envelope2* de la bibliothèque *chant-lib* – D'après le patch « envelope1+2.omp[168] ».

g) Les fonctions *ffreq, ampf* et *bdwth* : création de listes de paramètres pour les formants

Il a été vu précédemment que la fréquence, l'amplitude et la largeur de bande des différents formants du système vocal varient en fonction du type de voix et des voyelles utilisées. La bibliothèque *chant-lib* possède un ensemble de fonctions capables de fournir les paramètres des formants pour une synthèse FOF comme le montre la figure 65. Leurs paramètres sont les mêmes :

- *sexe* : type de voix utilisée (soprano, alto, contretenor, tenor ou basse). Il est ici important de rappeler que le noyau LISP d'OpenMusic utilise le protocole ASCII[169].

168 Disponible sur le cd à cd/om.

169 American Standard Code for Information Interchange.

Les noms des types de voix ne doivent donc pas contenir d'accents. La valeur par défaut de cet argument est « soprano ».

- *txt* : liste des voyelles à chanter (*a*, *e*, *i*, *o* ou *u*), le nombre d'éléments de cette liste doit être égal à celui de la liste des fréquences f_0, des onsets et des durées de notes. La valeur par défaut de cette argument est `(a)`.

Les fonctions permettant d'effectuer ces tâches sont donc :

- *ffreq* qui permet de délivrer une liste de listes de fréquences de formants. On l'utilise en la connectant sur l'enrée *ffreq* de la fonction *chant*.
- *ampf* donne une liste de listes d'amplitude de formants. On l'utilise en la connectant sur l'entrée *ampf* de la fonction *chant*.
- *bdwth* donne une liste de listes de largeurs de bandes de formants. On l'utilise en la connectant sur l'entrée *bdwth* de la fonction *chant*.

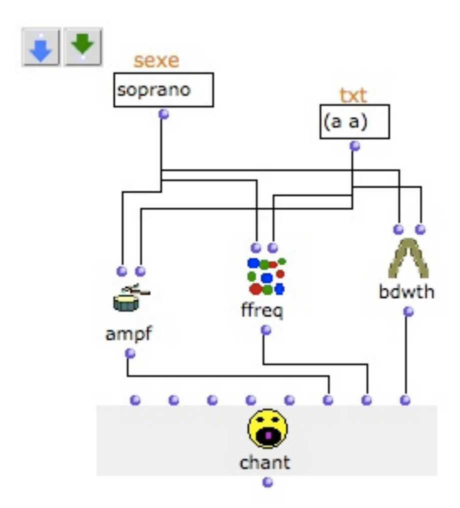


Fig. 65 – Créations de listes de paramètres de formants avec la bibliothèque *chant-lib* - D'après le patch « ampf+ffreq+bdwth.omp[170] ».

Le fonctionnement des fonctions *ffreq*, *ampf* et *bdwth* est décrit dans l'annexe n°17. Le traitement des informations retournées par *ffreq* et *bdwth* par la fonction *chant* est expliqué dans l'annexe n°18. Le cas de la fonction *ampf* est traité avec celui de la fonction *amp-cor* dans l'annexe n°21 puisque le traitement des informations par *chant* dépend ici aussi de la fonction *amp-cor*. Le traitement des largeurs de bandes, fréquences et amplitudes des formants dans CSOUND est présenté dans l'annexe n°19.

170 Disponible sur le cd à cd/om.

h) Les règles *auto-bw, auto-amp* et *auto-bend* de la bibliothèque *chant-lib*

Comme cela a été expliqué précédemment, les règles utilisées dans la bibliothèque *chant-lib* sont une simple ré-implémentation de celles présentes dans la bibliothèque *PW-Chant* de PatchWork dans la mesure où leur code LISP est totalement compatible avec celui d'OpenMusic. En effet, les seules modifications effectuées concernent la plupart du temps la forme des arguments pris par les fonctions afin de rendre ces dernières compatibles avec les autres fonctions de la bibliothèque *chant-lib*. Ainsi, les arguments concernant les fréquences de la fondamentale, les fréquences, les amplitudes et les largeurs de bandes des formants sont des listes de listes plutôt que de simples listes.

Il est important de rappeler que les algorithmes de chacune des fonctions qui vont être ici présentées ont été commentés dans (II)-(C). Il ne sera donc pas question ici de décrire leur code LISP comme cela a été fait pour les fonctions précédentes.

Enfin, il faut préciser que la règle *amp-cor* fera ici figure d'exception. Elle a en effet fait l'objet d'une réécriture complète dans CSOUND principalement à cause de la dimension dynamique de l'amplitude des formants qui varie en fonction de leur enveloppe.

o La fonction *auto-bw* et le calcul automatique de la largeur de bande des formants

Comme nous l'avons vu dans (II)-(C)-(e), il est possible de calculer la largeur de bande des formants de la voix en fonctions de leur fréquence grâce à une courbe parabolique définie par trois points de référence. Cette opération est effectuée dans la bibliothèque *chant-lib* par la fonction *auto-bw*. L'unique entrée (*freq*) de cette fonction prend donc en argument une liste de listes de fréquences de formants du même type que celle produite par les fonctions *ffreq* et *auto-bend*. Sa valeur par défaut est une liste de fréquence d'une soprano chantant un *a* : `((800 1150 2900 3900 4950))`. La fonction *auto-bw* peut par conséquent jouer le même rôle que la fonction *bdwth* et peut la remplacer comme le montre la figure 66.

o La fonction *auto-amp* et le calcul automatique de l'amplitude des formants

Il a été montré dans (II)-(C)-(f) qu'il est possible de calculer les amplitudes des formants de la voix en fonction de leur fréquence et de leur largeur de bande en simulant une

série de filtres. La fonction *auto-amp* de la bibliothèque *chant-lib* permet de mener à bien cette opération. Ses deux entrées prennent respectivement en argument des listes de listes des fréquences des formants (*freq*) et de leur largeur de bande (*bw*). Il est donc possible de connecter les fonctions *auto-bend* et *ffreq* sur l'entrée *freq* et les fonctions *auto-bw* et *bdwth* sur l'entrée *bw* comme le montre la figure 66. Les valeurs par défaut de ces deux entrées sont respectivement `((800 1150 2900 3900 4950))` et `((80 90 120 130 140))` ce qui correspond à une soprano chantant sur la voyelle *a*.

o La règle *auto-bend* et l'ajustement automatique de la fréquence du premier et du deuxième formant

Il a été expliqué précédemment que dans le cas de notes situées dans le registre aigu, la fréquence des deux premiers formants est modifiée, suivant alors les mouvements de la fondamentale f_0 lorsque celle-ci monte et pourrait dépasser la fréquence du premier formant. Cette règle est appliquée dans OpenMusic grâce à la fonction *auto-bend* de la bibliothèque *chant-lib* comme le montre la figure 66. Celle-ci prend donc en paramètre (dans l'ordre des entrées de l'objet) :

- *ffreq* : la liste de listes de fréquences des formants à corriger. Il est possible de connecter la fonction *ffreq* sur cette entrée. Sa valeur par défaut est `((800 1150 2900 3900 4950))` ce qui correspond à une soprano chantant sur la voyelle « a ».
- *sexe* : le type de voix (soprano, alto, contretenor, tenor, basse). La valeur par défaut de cette entrée est *soprano*.
- *f0* : la liste des fréquences de la fondamentale. Sa valeur par défaut est `(440)`.
- *corr* : le coefficient de correction de la fréquence des deux premiers formants. Sa valeur doit être supérieure ou égale à 1 (valeur par défaut) sinon une correction négative sera appliquée.

i) Le cas particulier de la fonction *amp-cor* pour l'ajustement de l'amplitude des formants en fonction de l'effort vocal

La fonction *amp-cor* de la bibliothèque *chant-lib* permet d'ajuster l'amplitude des formants produits lors de la synthèse avec l'objet *chant* en fonction de l'effort vocal. En effet, il a été montré dans (II)-(C)-(g) que le spectre d'un son de type vocal évolue en fonction de l'amplitude de la note chantée. Les paramètres nécessaires au fonctionnement de cette fonction sont donc (dans l'ordre des entrées de l'objet) :

- *ampf* : la liste de listes des amplitudes des formants à corriger. Sa valeur par défaut est `((-6 -32 -20 -50))` : une soprano chantant sur la voyelle *a*.
- *cslope* : l'indice d'inclinaison du spectre. Sa valeur par défaut est 1.
- *ajus* : la liste des coefficients d'ajustement influant sur l'une des deux formules de mise à l'échelle (*tino* et *scaler* dans l'algorithme présenté dans (II)-(C)-(g)). La valeur par défaut de cette entrée est `(1 1 0)`.
- *Sexe* : le type de voix (soprano, alto, contretenor, tenor, basse). La valeur par défaut est « `soprano` ».

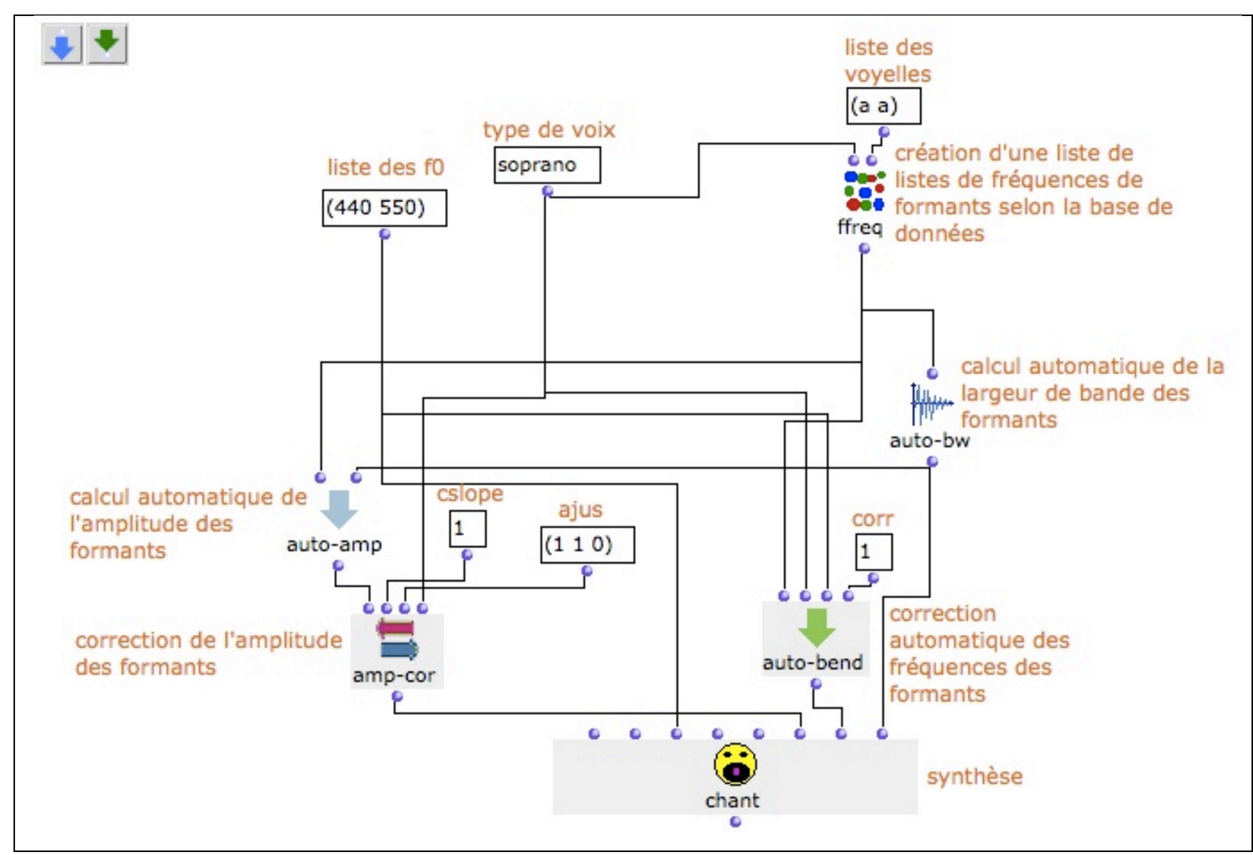


Fig. 66 – Calcul automatiques des paramètres pour la synthèse avec la bibliothèque *chant-lib* - D'après le patch « règles.omp[171] ».

Le traitement des informations envoyées à *amp-cor* est décrit dans l'annexe n°20.

Le traitement des informations retournées par les fonctions *amp-cor* et *ampf* par la fonction *chant* est expliqué dans l'annexe n°21.

La correction dans amplitudes des formants dans CSOUND à partir des paramètres issus de la fonction *amp-cor* est décrite dans l'annexe n°22.

171 Disponible sur le cd à cd/om.

j) La fonction *ampmid2db* et la conversion des amplitudes Midi en décibels

Comme nous l'avons expliqué en détail dans la partie (III)-(B)-(i), la bibliothèque *chant-lib* a été construite pour être au maximum compatible avec le standard Midi. Toutefois, l'utilisation des vélocités de notes Midi pour contrôler l'amplitude des sons produits avec la fonction *chant* présente quelques ambigüités auxquelles il est possible de remédier avec la fonction *ampmid2db*.

En effet, la vélocité d'une note Midi est exprimée sur une échelle linéaire dont les valeurs sont des entiers naturels compris entre 0 et 127 tandis que la liste des amplitudes données à la fonction *chant* doit être exprimée en décibels (échelle logarithmique). Afin de convertir des dynamiques Midi en décibels il est donc nécessaire d'établir une relation entre amplitude et vélocité de notes Midi.

L'ensemble des tests qui vont êtres présentés par la suite ont été effectués à partir du même fichier Midi « test-dynamique.mid »[172] afin de corréler les résultats obtenus. Ce dernier a été créé dans OpenMusic à l'aide du patch « test-dynamique.omp »[173] qui génère une série de 127 notes de même fréquence à intervalles réguliers avec une amplitude croissante allant de 0 à 127 avec un pas de un :

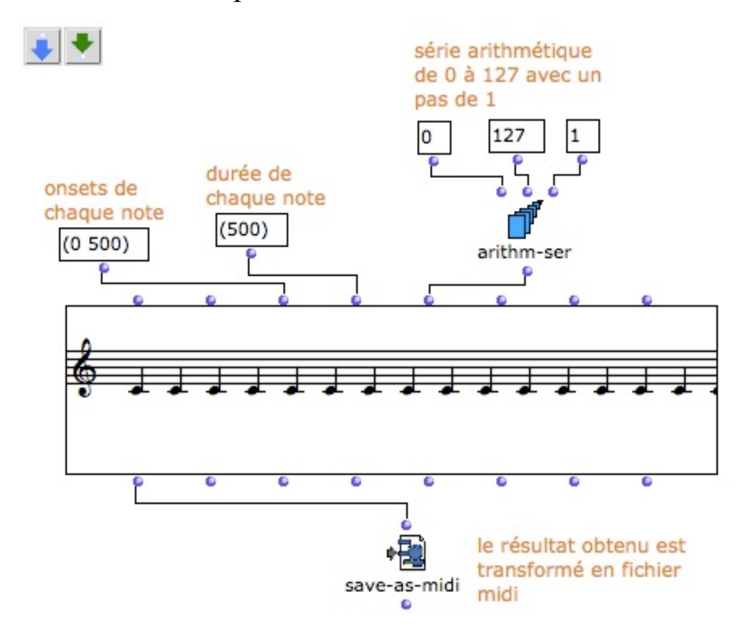


Fig. 67 – Crescendo augmentant la vélocité Midi de un à chaque note – D'après le patch « test-dynamique.omp ».

---

172 Disponible sur le cd à cd/test-dynamique.

173 Ibid.

Le premier test effectué a permis d'analyser le comportement d'OpenMusic vis à vis des dynamiques Midi. La classe *chord-seq* permet en effet de connaître la « nuance musicale » de chaque note qu'elle contient comme le montre la figure suivante :

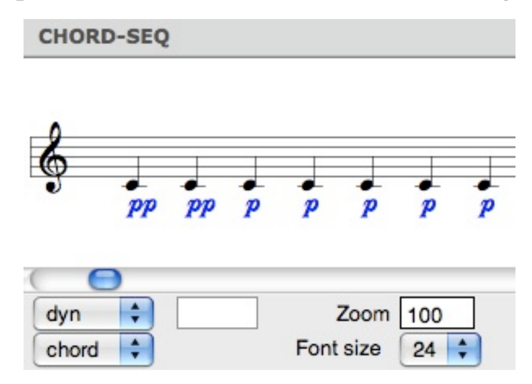


Fig. 68 – Affichage des nuances musicales dans la classe *chord-seq* d'OpenMusic.

Les travaux de Fletcher et Munson[174] ont permis de montrer qu'il existe des correspondances entre les nuances musicales et les amplitudes en décibels :

Nuances musicales	**Abréviations**	**Amplitudes en décibels**
Pianississimo	ppp	40
Pianissimo	pp	50
Piano	p	60
Mezzo-piano	mp	66
Mezzo-forte	mf	76
Forte	f	80
Fortissimo	ff	90
Fortississimo	Fff	100

Fig. 69 – Tableau des correspondances entre nuances musicales et amplitudes en décibels d'après Fletcher et Munson.[175]

Il est donc possible d'associer une vélocité Midi à une amplitude en décibels en se basant sur les valeurs données dans OpenMusic :

174 FLETCHER, Harvey ; MUNSON, William, « Loundness, its definition, measurement and calculation », *Journal of the Acoustical Society of America*, V (1933), n° 2, p. 82-108.

175 http://www.aamhl.org/dBchart.htm (en ligne le 04/09/2010).

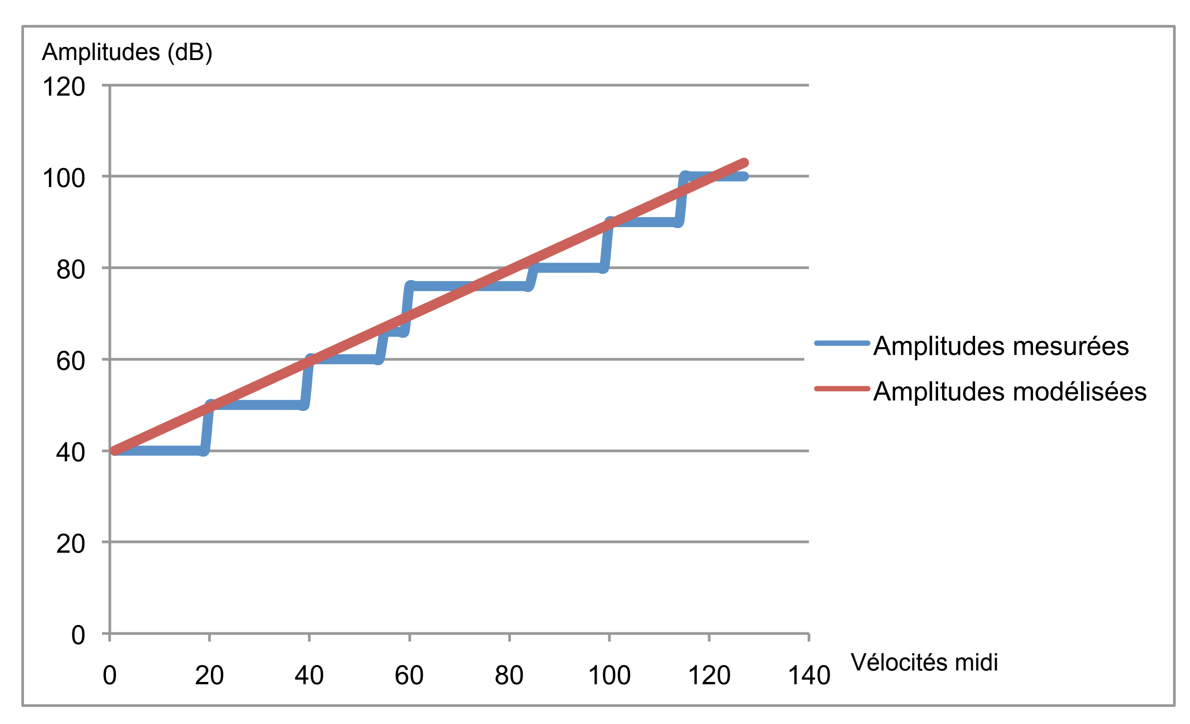


Fig. 70 – Conversion des vélocités Midi en amplitudes en décibels dans OpenMusic.[176]

Dans le cas présent, on peut voir que l'amplitude évolue de façon quasi linéaire (courbe bleue). Il est donc possible de modéliser ce comportement (courbe rouge) avec la formule suivante :

AmplitudeDécibels = (*DynamiqueMidi* x 0,5) + 40

Pour effectuer le test suivant, le fichier « test-dynamique.mid » a été joué au piano avec le synthétiseur Midi de Mac OS X. Le son normalisé obtenu a été enregistré dans le fichier « test-dynamique.aif »[177]. L'évolution de l'amplitude du son enregistré au cours du temps a alors été mesurée dans Max/MSP avec le patch « test-dynamique.maxpat »[178]. Ce dernier utilise l'objet *fiddle* qui permet d'effectuer tout un ensemble de mesures sur des signaux sonores. Les amplitudes, qui peuvent être mesurées en décibels ou en raw sont alors affichées dans la fenêtre de message de Max/MSP. Afin de réduire le nombre d'informations à traiter, les mesures sont effectuées toutes les 250ms, soit environ deux fois par notes (*DuréeNote* = 500ms). Enfin, la valeur maximale atteinte par l'amplitude en décibels et l'amplitude raw est enregistrée.

---

176 Le tableau contenant les valeurs qui ont permis le traçage de ce diagramme est disponible sur le cd dans le fichier test-dynamique.xslx à cd/test-dynamique.

177 Disponible sur le cd à cd/test-dynamique.

178 Ibid.

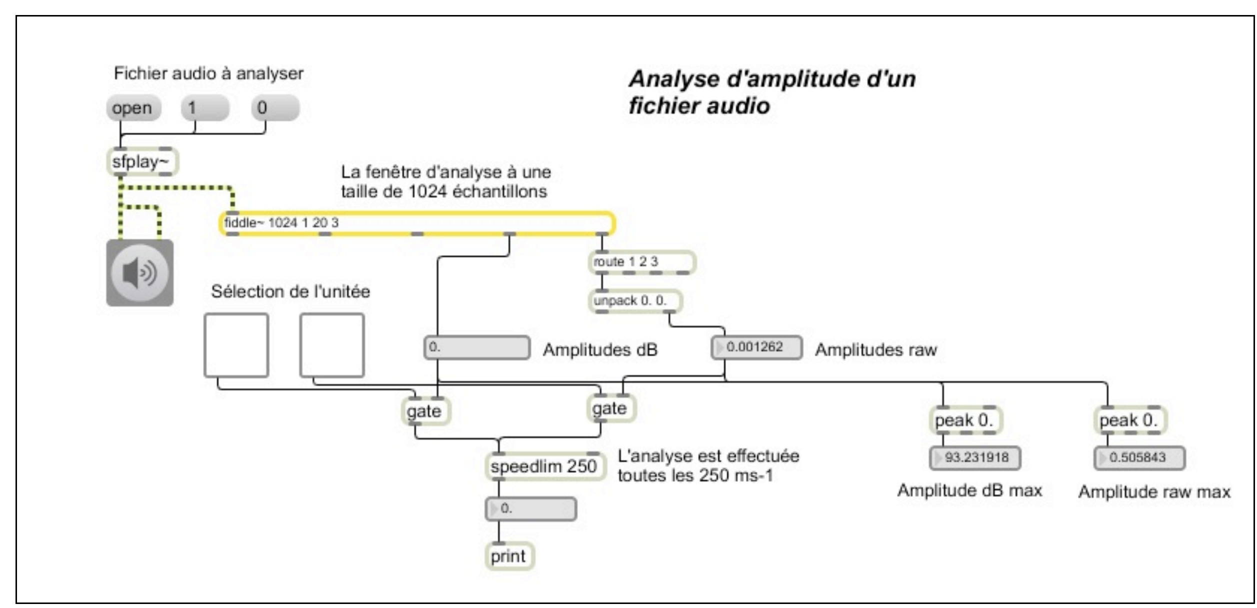


Fig. 71 – Patch Max/MSP pour l'analyse dynamique de l'amplitude de fichiers audio en décibels et en raw – D'après le patch « test-dynamique.maxpat ».

L'analyse du fichier « test-dynamique.aif » par le patch Max/MSP présenté précédemment a fourni un total de 248 échantillons d'amplitudes en raw et en décibels. L'ensemble de ces mesures ont été placées dans un tableau de la forme :

Numéro de l'échantillon	Amplitudes Midi calculées d'après le numéro de l'échantillon	Amplitudes mesurées en raw dans Max/MSP	Amplitudes mesurées en dB dans Max/MSP	Amplitudes raw converties linéairement en dB
1	0,512096774	0,001266	20,194746	0,235476
2	1,024193548	0,001266	20,096311	0,235476
3	1,536290323	0,001266	27,746653	0,235476
n	(*NumEch$_n$* x 127)/248	*AmpRaw$_n$*	*AmpdB$_n$*	AmpRaw$_n$ x 186

Fig. 72 – Tableau comparatif des amplitudes calculées dans différentes unités à partir du fichier « test-dynamique.aif ».

Il est possible de consulter le tableau complet dans le CD dans le fichier « test-dynamique.xslx » à cd/test-dynamique.

Les dynamiques Midi ont été déduites du numéro de chaque échantillon en effectuant l'opération suivante : *AmplitudeMidi* = (*NuméroEchantillon* x 127)/248. Dans la mesure où le but de ces analyses est d'établir une relation schématique entre vélocité et amplitude de notes Midi, il n'est pas nécessaire d'avoir des mesures précises à l'échantillon près, une certaine marge d'erreur est donc tolérable. Il est alors possible de simplifier la formule précédente : *AmplitudeMidi* = *NuméroEchantillon* / 2.

Afin de modéliser le comportement de la courbe des amplitudes raw pour une utilisation avec la bibliothèque *chant-lib*, il est nécessaire de convertir linéairement ces dernières en décibels. Pour ceci, l'amplitude maximale contenue dans le fichier « test-dynamique.aif » a été relevée en décibels (environ 93dB) et en raw (environ 0.5) et a permis

d'établir un ratio entre ces deux valeurs : 186,1 . Ce dernier est utilisé pour mettre à l'échelle les valeurs raw en décibels.

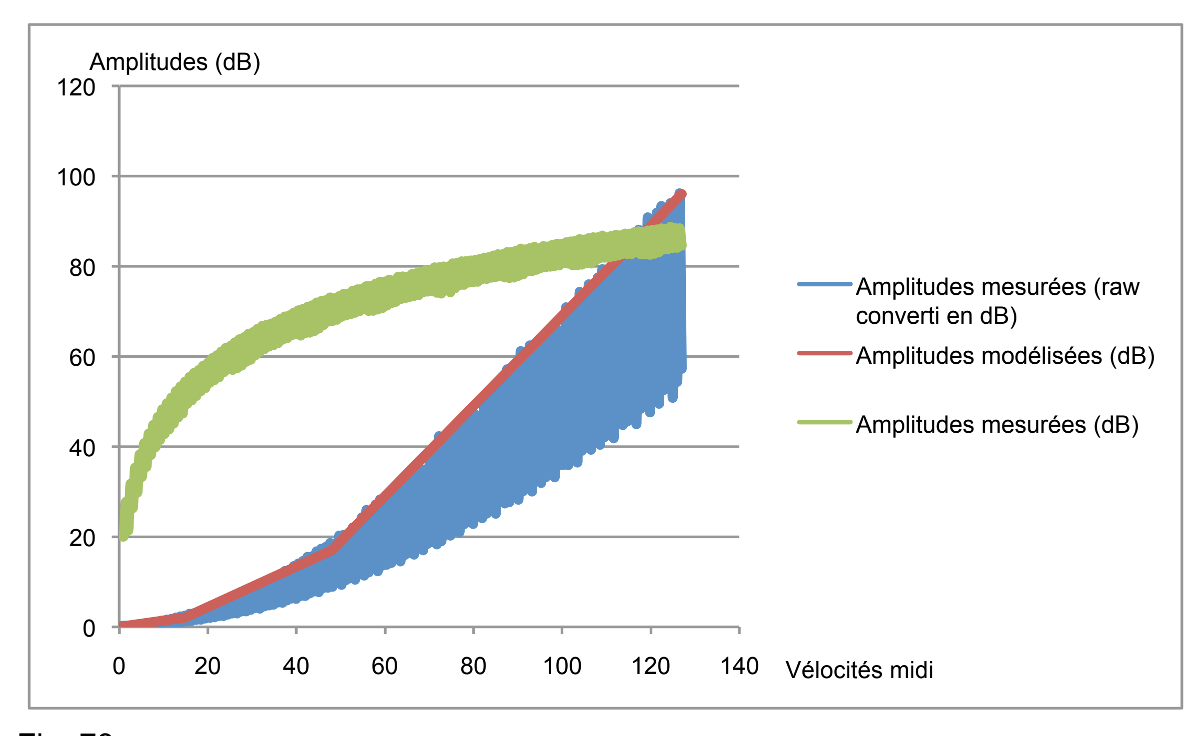


Fig. 73 – Evolution de l'amplitude du fichier « test-dynamique.aif » mesurée en décibels et en raw (ici converti linéairement en décibels avec la formule : AmplitudeRaw x 186) en fonction de la vélocité Midi. La courbe rouge est une modélisation en trois phases de l'évolution de l'amplitude raw. Ce modèle sera utilisé dans OpenMusic avec la fonction *ampmid2db* pour convertir des vélocités midi en amplitude en décibels.[179]

Il est facile de constater que la courbe la plus semblable à celle présentée dans la figure 70 est celle des amplitudes mesurées en raw converties linéairement en décibels, la courbes des amplitudes mesurées en décibels ayant une forme logarithmique. Par conséquent, le modèle utilisé dans la fonction *ampmid2db* est celui des amplitudes mesurées raw.

Cette dernière a une évolution en trois phases quasi linéaires. Les deux premières ont lieu en dessous de 20dB et la dernière au dessus ce qui correspond à une vélocité Midi d'environ cinquante. Cette information est importante à souligner. En effet, les sons en dessous de 20dB sont difficilement audibles. Ainsi, un modèle simplifié de cette courbe pourrait considérer que toute vélocité Midi inférieure à cinquante correspond à une valeur de 0dB et que toute vélocité Midi supérieure à cinquante correspond à une amplitude dont la valeur peut être calculée linéairement en fonction de la vélocité.

179 Le tableau contenant les valeurs qui ont permis le traçage de ce diagramme est disponible sur le cd dans le fichier test-dynamique.xslx à cd/test-dynamique.

Le modèle utilisé dans *ampmid2db* est légèrement différent puisqu'il prend aussi en compte les deux premières phases de l'évolution de la courbe des amplitudes raw converties en décibels. Il est de la forme (courbe rouge de la figure 73) :

- *AmplitudeDécibels* = *DynamiqueMidi* x 0,14 pour 0≤*DynamiqueMidi*<14
- *AmplitudeDécibels* = (*DynamiqueMidi* x 0,45) – 4,61 pour 14≤*DynamiqueMidi*<48
- *AmplitudeDécibels* = *DynamiqueMidi* – 31 pour 48≤*DynamiqueMidi*

Le fonctionnement de la fonction *ampmid2db* est décrit dans l'annexe n°23.

k) Les autres fonctions de *chant-lib*

Les autres fonctions encore non présentées de la bibliothèque *chant-lib* sont principalement utilisées pour effectuer des calculs de conversion ou des opération très basiques sur les listes d'onsets et de voyelles afin de faciliter l'utilisation du système.

La fonction *kill-last*

La fonction *kill-last* de la bibliothèque *chant-lib* permet de supprimer le dernier élément d'une liste. Elle peut-être très utile lors de l'utilisation d'une classe *chord-seq* d'OpenMusic pour supprimer un silence à la fin d'une phrase musicale en modifiant la liste d'onsets. Sa seule entrée correspond donc la liste à modifier (*duree*) (cf. figure 74).

Pour accomplir cette tâche, une boucle copiant le contenu d'une liste dans une autre et ne lisant pas le dernier élément de cette liste est utilisée :

```
(om::defmethod! kill-last ((duree list))
  :icon 178
  :indoc '("liste de durees")
  :initvals '(0)
  (let ((lduree (length duree)) res)
    (dotimes (n (- lduree 1) res)
             (push (nth n duree) res))
    (reverse res)))
```

La fonction *onset-calc*

La fonction *onset-calc* de la bibliothèque *chant-lib* permet de déduire une liste d'onset à partir d'une liste de durées. Cela peut permettre du supprimer tout les silences d'une phrase musicale qui peuvent parfois êtres gênants ou non souhaités en particulier si la source est un fichier Midi. La seule entrée de cette fonction (*duree*) prend donc en argument une liste de durée (cf. figure 74).

Pour mener à bien cette opération, *onset-calc* crée une liste commençant par zéro et dont les autres éléments sont calculés selon la formule suivante :

$onset_n = durée_{n-1} + durée_n$.

```
(om::defmethod! onset-calc ((duree list))
  :icon 178
  :indoc '("dures des notes")
  :initvals '(0)
  (let ((lduree (length duree)) res temp (hold 0))
       (push 0 res)
       (dotimes (n (- lduree 1) res)
                (setf temp (+ (nth n duree) hold))
                (setf hold temp)
                (push temp res))
       (reverse res)))
```

- **La fonction *voy-calc***

Dans le cas où l'on souhaiterait synthétiser une phrase musicale en utilisant toujours la même voyelle, il peut être intéressant d'avoir une fonction créant une liste où la même lettre est répétée en fonction du nombre de notes à jouer. La fonction *voy-calc* de la bibliothèque *chant-lib* permet d'effectuer cette opération (cf. figure 74).

Sa première entrée prend en argument la liste des fréquences de la fondamentale et sa deuxième la voyelle à répéter (la lettre ici donnée n'est pas considérée comme un caractère, il n'est donc pas nécessaire de l'encadrer avec des guillemets) :

```
(om::defmethod! voy-calc ((f0 list) (txt t))
  :icon 178
  :indoc '("note" "voyelle")
  :initvals '('(0) 'a)
  (make-list (length f0) :initial-element txt))
```

I) Interfaçage des objets de la bibliothèque *chant-lib* avec la classe *chord-seq*

La bibliothèque *chant-lib* a été construite pour avoir une grande compatibilité avec les autres fonctions d'OpenMusic. Ainsi, il peut par exemple être intéressant d'utiliser la classe *chord-seq*[180] pour piloter la fonction *chant* de façon intuitive. Celle-ci permet d'écrire des partitions dans OpenMusic et de les connecter à d'autres objets. Elle est basée sur le standard Midi.

Ses cinq premières entrées permettent de l'interfacer avec les différentes fonctions de la bibliothèque *chant-lib*, elles sont dans l'ordre :

180 HADDAD, Karim, *OpenMusic User's Manual*, Paris : IRCAM, 2003.

- *self* : l'objet lui-même
- *lmidic* : liste des notes en midicents
- *lonset* : durée des onsets en millisecondes
- *ldur* : durée des notes en millisecondes
- *lvel* : amplitude des notes en Midi.

Les fonctions de la bibliothèque *chant-lib* fonctionnant en hertz pour les fréquences de la fondamentale, il est nécessaire de convertir dans cette unité les notes fournies par *lmidic*. Pour ceci, il est possible d'utiliser la fonction *mc->f*.

Les données fournies par les entrées *lonset* et *ldur* doivent elles aussi faire l'objet d'une conversion qui reste néanmoins très simple : il suffit de les diviser par mille.

Enfin, il est nécessaire d'utiliser la fonction *ampmid2db* pour rendre les données fournies par *lvel* compatibles avec l'ensemble des fonctions de la bibliothèque *chant-lib* (cf. (III)-(B)-(h)).

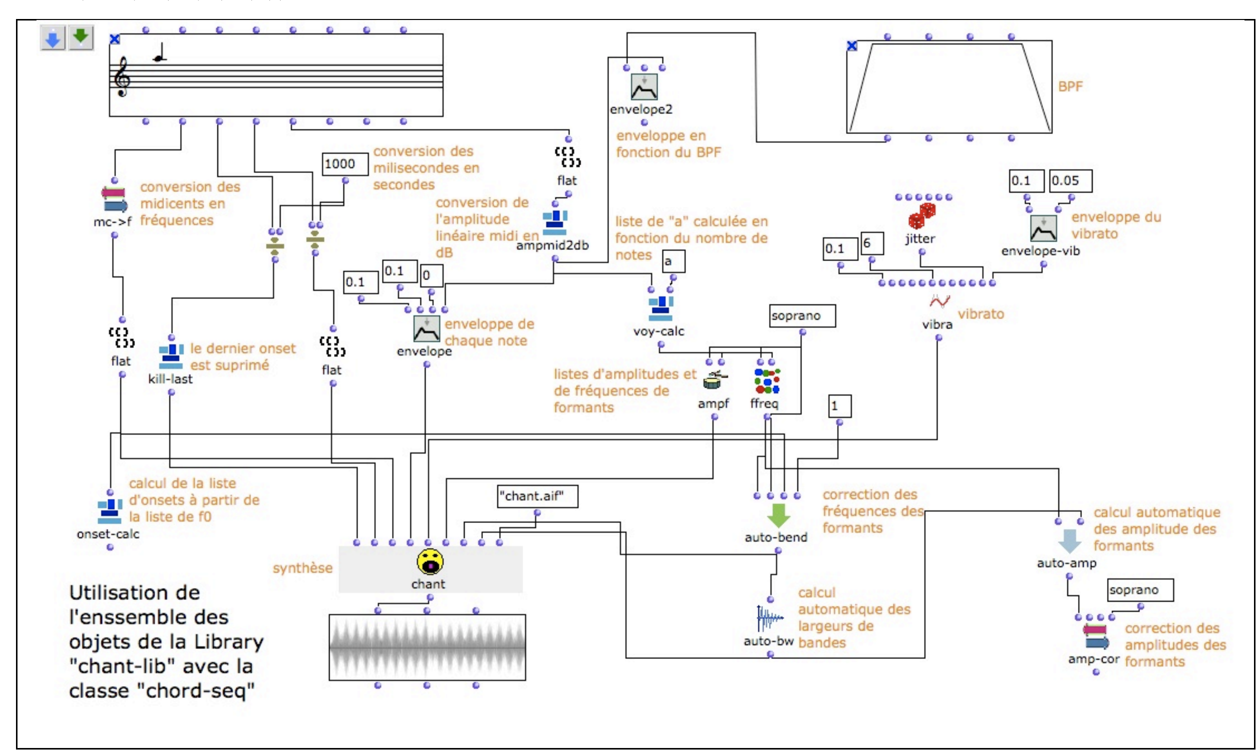


Fig. 74 – Utilisation de l'ensemble des fonctions de la bibliothèque *chant-lib* avec la classe *chord-seq* - d'après le patch « exemple-complet.omp[181] ».

181 Disponible sur le cd à cd/om.

C. Test des performances de la bibliothèque *chant-lib* pour OpenMusic à travers l'exemple de *L'air de la reine de la nuit* de *La flûte enchantée* de Mozart

En 1984, dans le but de démontrer l'efficacité de la synthèse par Fonctions d'onde formantique et du programme CHANT, Xavier Rodet et son équipe ont travaillé sur un projet de synthèse d'un extrait de *L'air de la reine de la nuit* de *La flûte enchantée* de Mozart qu'il est possible d'entendre dans « reine-IRCAM.aiff[182] ». Bien que le texte d'origine ne soit pas utilisé et que l'ensemble de l'air soit chanté sur la voyelle *a*, le résultat obtenu fut tellement impressionnant qu'il est devenu l'une des rares expériences de synthèse pure de la voix chantée connue du grand public. Il fut par exemple cité comme exemple de référence par Max Mathews lors d'une interview télévisée en 1986 afin de montrer les progrès fait par l'informatique musicale depuis son apparition dans les années 1950.[183]

Afin de démontrer l'efficacité de la bibliothèque *chant-lib* pour OpenMusic dans la ré-implémentation du programme CHANT, il peut être intéressant de tenter de reproduire le travail effectué par l'équipe de Xavier Rodet en 1984. Dans cette partie, la démarche suivie pour effectuer ce travail est présentée.

a) Analyse du fichier audio original

Bien que *L'air de la reine de la nuit* synthétisé par l'équipe de Xavier Rodet soit très populaire, aucune documentation n'a été publiée au sujet de sa création. Il est donc nécessaire de procéder à une analyse détaillée du fichier audio si on souhaite reproduire les paramètres de la synthèse de la façon la plus précise possible. Cette analyse passe principalement par l'étude du spectrogramme généré avec le programme AudioSculpt à partir du fichier audio.

La voix étant accompagnée par un piano acoustique, il est assez aisé de distinguer ses partiels de ceux de l'instrument. En effet, les propriétés spectrales d'une voix chantée sont très différentes de celle d'un piano qui n'est par exemple pas en mesure de

---

182 Disponible sur le cd à cd/audio.

183 Vidéo disponible sur le cd dans le fichier « Mathews-reine.m4v » à cd/vidéo et sur le site YouTube à l'adresse suivante : http://www.youtube.com/watch?v=_15ZQL82P4M (en ligne le 04/09/10).

produire un vibrato et dont l'amplitude des notes décroit au cours du temps comme le montre la figure 75.

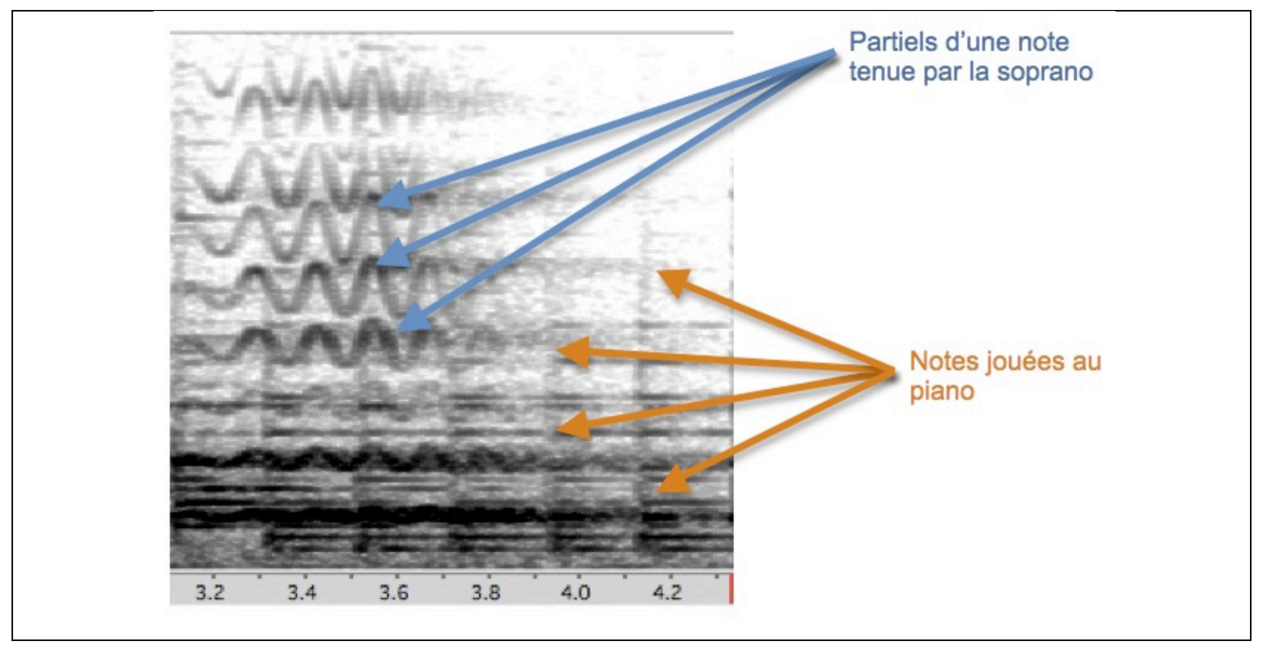


Fig. 75 – Spectrogramme d'une note tenue par la soprano de synthèse accompagnée au piano.

L'analyse de la durée de la période du vibrato en plusieurs points du fichier audio à permis de déduire sa fréquence moyenne qui est donc d'environ 6,5 Hz d'après le calcul suivant : *fréquence moyenne* = 1/*période moyenne* où *période moyenne* ≈ 0,150 secondes. L'analyse du spectrogramme dans son ensemble a également permis de déduire que l'interpolation linéaire des formants n'est jamais utilisée et qu'une enveloppe d'amplitude de type attaque – maintien – chute est appliquée à l'ensemble des notes comme le montre la figure 76.

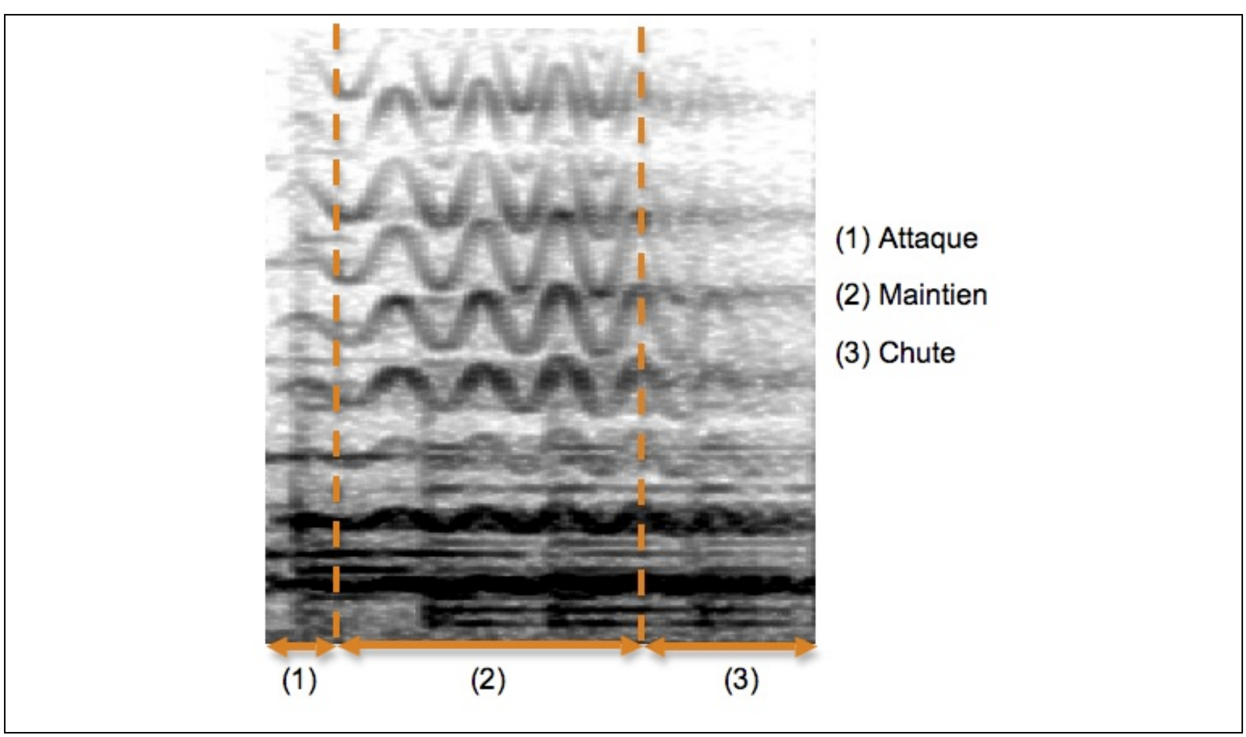


Fig. 76 – Enveloppe d'amplitude d'une note tenue par la soprano de synthèse.

Il a été remarqué que deux grands types de paramètres pour la synthèse ont été utilisés dans l'ensemble de la partie ici concernée de *L'air de la reine de la nuit* :

Paramètres pour des notes aigües piquées de courte durée :

- L'attaque et la chute de l'enveloppe d'amplitude de chaque note ont une durée très courte : en moyenne, respectivement 0,04 et 0,06 secondes.
- Aucune enveloppe d'amplitude n'est utilisée pour le vibrato.
- L'amplitude globale du vibrato est très faible (entre 0,01 et 0,05). Ce point est important à souligner. En effet, la durée des notes dans ce cas là (en moyenne 0,156 secondes) est inférieure à la période du vibrato (environ 0,167 secondes). Ainsi, si l'amplitude du vibrato était ici trop importante, les notes seraient probablement désaccordées. Il est néanmoins nécessaire de conserver une modulation de la fréquence de la fondamentale afin que le son produit garde des propriétés « vocales ».

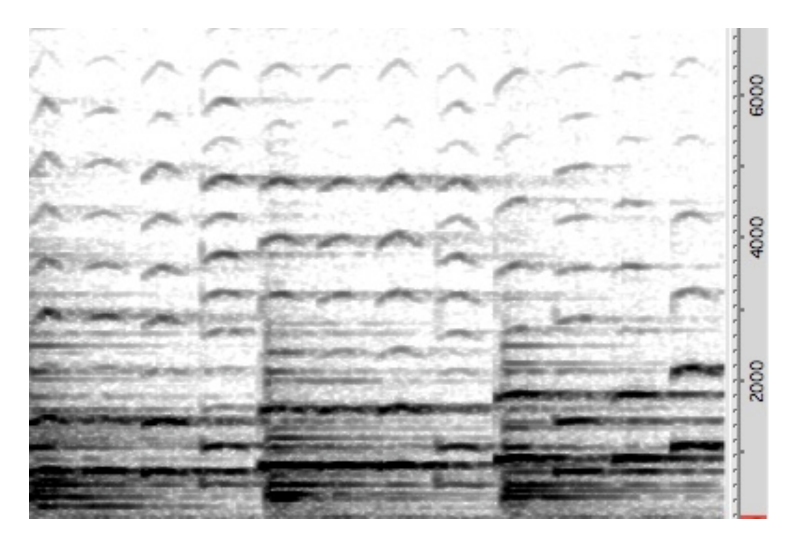


Fig. 77 – Spectrogramme d’une succession de 12 notes piquées de courte durée chantées par la soprano de synthèse.

Paramètres pour des notes longues du milieu de la tessiture d’une soprano :

- L’attaque est plus courte que la chute : en moyenne, respectivement 0,1 et 0,2 secondes. Ceci crée un effet d’atténuation de l’amplitude à la fin de chaque note.
- Une enveloppe d’amplitude est utilisée pour le vibrato. Ces paramètres sont semblables à ceux de l’enveloppe décrite précédemment.
- Le vibrato, assez ample à une amplitude globale d’environ 0,1

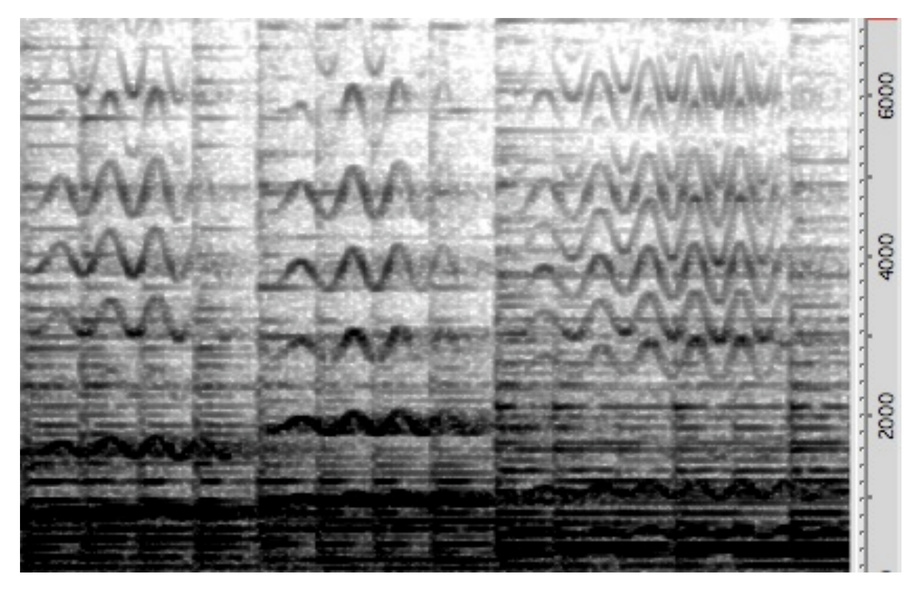


Fig. 78 – Spectrogramme de trois notes successives tenues chantées par la soprano de synthèse.

b) Synthèse de *L’air de la reine de la nuit* à l’aide d’OpenMusic

OpenMusic est une plateforme de travail qui permet d'interconnecter un grand nombre de procédés et de standards. Dans notre cas, il est donc important d'exploiter au maximum les possibilités offertes. Ainsi, il est possible dans OM d'utiliser un fichier Midi comme source à une série d'événements dans une classe *chord-seq* par exemple. On peut alors connecter les classes *midifile* et *chord-seq* entre elles comme le montre la figure 79.

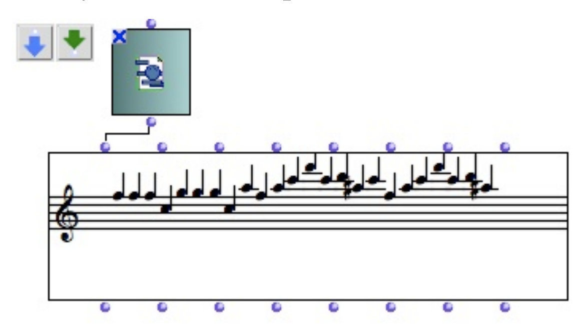


Fig. 79 – Exemple d'utilisation d'un fichier Midi dans OpenMusic – D'après le patch « midi.omp[184] ».

Il a été montré précédemment que la classe *chord-seq* est compatible avec la bibliothèque *chant-lib*, on en déduit par récursivité que les fichiers Midi le sont aussi.

La première étape du travail de synthèse de *L'air de la reine de la nuit* est donc passée par la création d'un fichier Midi contenant la partie à chanter et l'accompagnement sur des pistes Midi séparées. Cette opération a été effectuée dans Cubase à partir du fichier reine-original.MID[185]. Ce dernier a fait l'objet de légères retouches en particulier au niveau des figures rythmiques qui ont été quantizées. Le résultat obtenu après cette étape est présenté dans le fichier reine-final.mid[186].

La partition de la partie de soprano a ensuite fait l'objet d'un découpage en quatorze sections correspondant chacune à un des deux cas de figures présentés précédemment dans l'étape d'analyse.[187] Chaque section a par la suite été convertie en fichier Midi utilisable dans OpenMusic.[188]

Un ensemble de quatorze patchs ont été créés dans OpenMusic afin de synthétiser le contenu des différents fichiers Midi[189] [190]. Tous ces patchs ont été construits sur la même base. Les seuls divergences visibles se trouvent au niveau de l'utilisation ou non de la fonction *envelope-vib* et de paramètres différents, conformément à ce qui a été expliqué

---

184 Disponible sur le cd à cd/om.

185 Disponible sur le cd à cd/reine. Fichier Midi téléchargé à l'adresse suivante : http://leonard.www.itaque.com/ (en ligne le 04/09/2010).

186 Disponible sur le cd à cd/reine. La partition correspondante à ce fichier Midi est disponible dans l'annexe n°25.

187 Le découpage effectué est visible dans la partition disponible dans l'annexe n°25.

188 L'ensemble des fichiers Midi créés sont disponible sur le CD à cd/reine/decoupage-midi.

189 L'ensemble des patchs sont disponibles sur le cd à cd/reine/om.

190 Les paramètres détaillés de chaque patchs sont résumés dans un tableau disponible dans l'annexe n°25.

dans la partie précédente sur l'analyse du fichier audio d'origine. Les patchs utilisés sont donc de la forme :

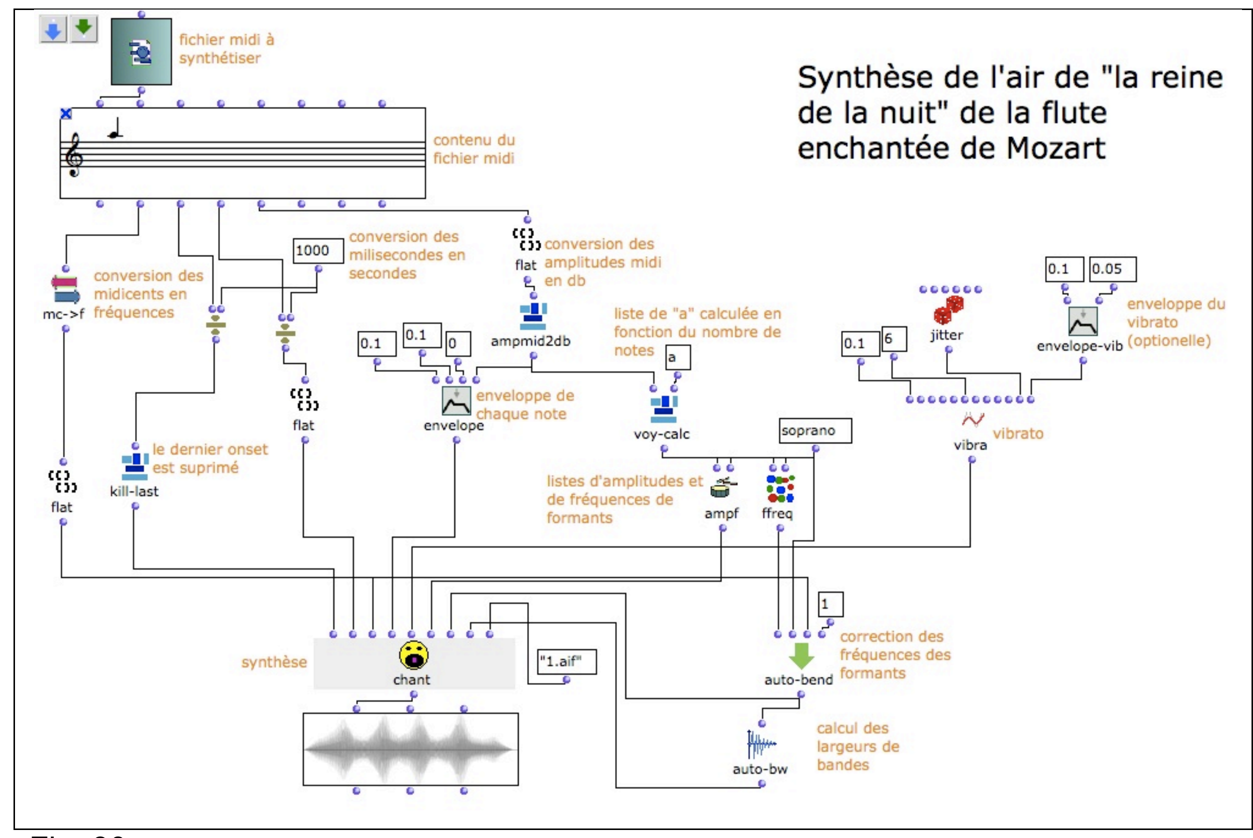


Fig. 80 – Patch OpenMusic pour la synthèse des sections des phrases musicales de *L'air de la reine de la nuit* de *La flute enchantée* de Mozart - d'après le patch « exemple-reine.omp[191] ».

On peut voir que les largeurs de bandes des formants sont calculées automatiquement et que la fréquence des deux premiers d'entre eux est ajustée en fonction de la fréquence f_0 de la fondamentale avec la fonction *auto-bend*. Ce dernier point est particulièrement important dans le cas présent. En effet, les notes à synthétiser ont des fréquences très élevées qui pourraient être supérieures à la fréquence des deux premiers formants.

Les quatorze fichiers audio créés[192] par les différents patchs ont ensuite été montés dans Cubase afin de reconstituer le passage musical synthétisé dans son intégralité. Cubase a également permis d'ajouter un accompagnement au piano. Ce dernier est joué avec le synthétiseur HALionOne intégré à la version 4 quatre de Cubase. Enfin, le mixage a été effectué lors de cette étape où une légère réverbération a été fut ajoutée à l'ensemble des

191 Disponible sur le cd à cd/om.

192 L'ensemble des fichiers audio créées sont disponibles sur le cd à cd/reine/découpage-audio.

pistes dans le but de rendre le résultat final plus naturel.[193] Il est possible d'écouter le résultat obtenu dans le fichier « reine-synth.aif[194] ».

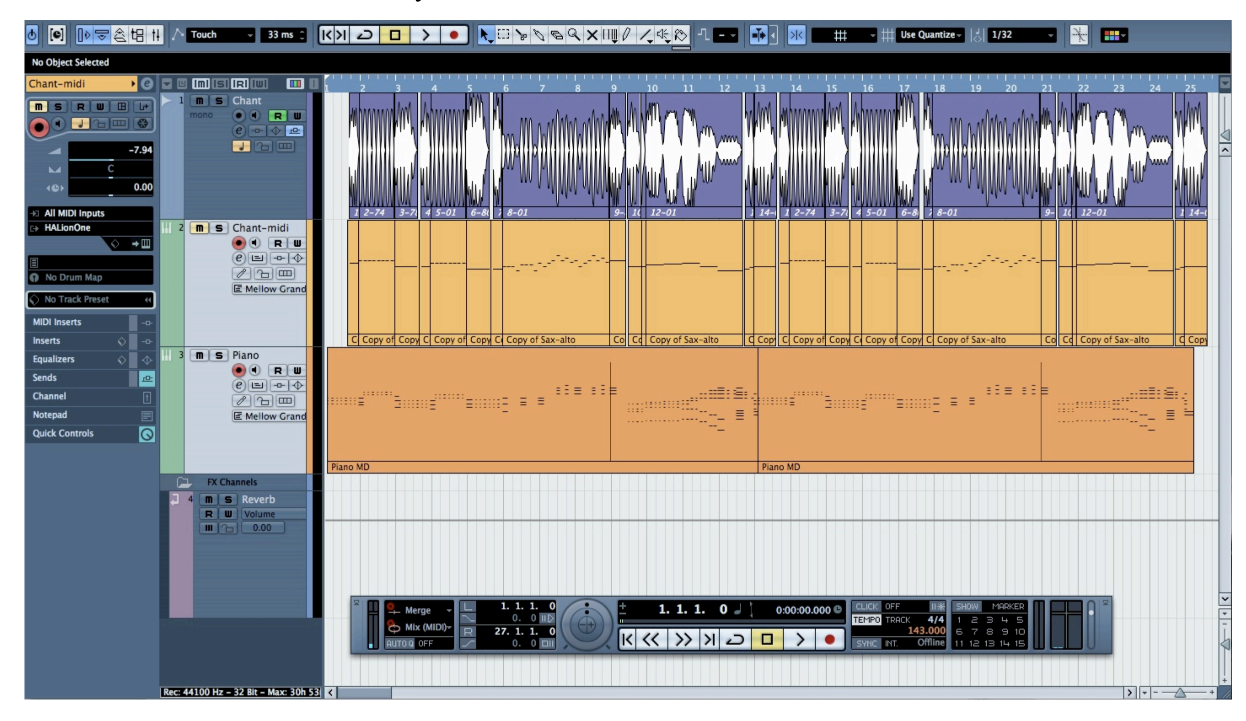


Fig. 81 – Montage dans Cubase des différents fichiers audio générés dans OpenMusic pour la synthèse de *L'air de la reine de la nuit* de *La flute enchantée* de Mozart.

L'implémentation du programme CHANT faite dans OpenMusic à travers la bibliothèque *chant-lib* permet l'acquisition de résultats tout à fait convainquants comme le montre l'exemple de synthèse de *L'air de la reine de la nuit* de *La flute enchantée* de Mozart.

L'adaptation du code LISP des règles de la bibliothèque *PW-chant* de PatchWork pour le rendre compatible avec OpenMusic a été une opération relativement aisée bien qu'un important nombre de modifications ont du y être apportées.

L'interfaçage de OpenMusic avec CSOUND est rendu particulièrement efficace et fonctionnel grâce à la bibliothèque *om2csound*. Enfin, il est important de noter que l'utilisation combinée de ces deux programmes s'avère être une solution intéressante pour la création rapide et efficace d'outils de synthèse de sons avec interface graphique.

193 Le fichier Cubase « rein.cpr » est disponible sur le cd à cd/reine/cubasse.

194 Disponible sur le cd à cd/reine.

Conclusion

La synthèse par fonctions d'onde formantique a été utilisée dans un nombre conséquent de composition musicale pour la synthèse de la voix chantée au cours des trente dernières années. La simplicité du procédé, le contrôle important sur les différents paramètres du son et la qualité des résultats obtenus pour la synthèse de sons de type voisé comme les voyelles ont attiré un grand nombre de compositeurs.

Le programme CHANT a été la principale plateforme d'utilisation de cette technique pour la composition musicale. Il offrait des possibilités qu'il est impossible de retrouver aujourd'hui dans les programmes disponibles sur le marché. L'implémentation des différentes fonctions de CHANT dans le programme OpenMusic interfacé avec CSOUND en s'inspirant de la bibliothèque *Chant-PW* autrefois disponible dans PatchWork s'est révélé être une solution possible adaptée au problème.

Dans le système présenté, OpenMusic permet un contrôle graphique ludique et intuitif des différents paramètres de la synthèse avec une collection d'objets gravitants autour d'une fonction principale (*chant*) permettant de regrouper l'ensemble des paramètres et de les envoyer au programme CSOUND. Ce dernier a permis d'effectuer les étapes de traitement du signal avec une très grande simplicité grâce à l'objet `fof`.

Les résultats obtenus lors de l'étape de test pour la synthèse de *L'air de la reine* de *La nuit de la flute enchantée* de Mozart sont tout à fait convenables compte tenu du nombre très restreint de paramètres utilisés.

Les principales règles de CHANT ont été intégrées dans différents objets regroupés dans la bibliothèque *chant-lib*. Toutefois, bien que l'ensemble des fonctions et des règles de CHANT y soient implémentées, certaines fonctions de *Chant-PW* n'ont pas été traduites et quelques éléments restent à améliorer notamment au niveau de la précision du contrôle des paramètres. Il serait par exemple intéressant de pouvoir modifier les valeurs par défaut des amplitudes, largeurs de bandes et fréquences des formants par l'intermédiaire de fichiers textes fournis aux objets *ampf*, *ffreq* et *bdwth*. Il serait aussi nécessaire d'intégrer un générateur de bruit blanc filtré (pour les sons vocaux de type fricatif) dont les paramètres seraient contrôlés en fonction des consonnes fournies au synthétiseur. Ces modifications pourraient faire l'objet de recherches ultérieures et pourraient être apportées facilement dans

la mesure où l'ensemble des programmes utilisés sont conçus sous une licence OpenSource GNU/GPL[195].

Depuis son apparition dans les années soixante, la synthèse numérique de la voix chantée est un domaine qui a connu une grande évolution. L'augmentation de la puissance de calcul des ordinateurs au cours de la seconde moitié du vingtième siècle a permis d'envisager l'implémentation de modèles de plus en plus complexes. Les modélisations de l'appareil phonatoire par des tubes acoustiques à une dimension ont laissé place à des techniques permettant l'acquisition de résultats toujours meilleurs. Aujourd'hui, il est possible de distinguer trois grands types de techniques pour la synthèse de la voix chantée.

Les modèles physiques de l'appareil phonatoire tentent de reproduire mathématiquement les différentes interactions entre chacun des organes qui aboutissent à la phonation. Plus un modèle physique est complet (et par conséquent complexe), plus il permet l'obtention de résultats proches du son naturel. L'avantage de ce type de technique est la flexibilité et la possibilité d'intervenir sur chaque paramètre du son. Toutefois, leur complexité, le grand nombre de paramètres nécessaires pour la synthèse et par conséquent la quantité importante de calculs impliqués en font des outils qui sont aujourd'hui peu utilisés au détriment de techniques plus simples à implémenter.

Les modèles spectraux tentent de reproduire un son naturel en se basant sur l'analyse spectrale de ce son. Plusieurs techniques peuvent êtres classées dans cette catégorie. La synthèse par modulation de fréquence permet d'obtenir d'excellents résultats avec un nombre de paramètres très restreints et est très peu coûteuse en calculs de par sa simplicité. De plus, elle permet d'obtenir des résultats de très bonne qualité pour la synthèse des sons apériodiques (consonnes). Les techniques de synthèse additive visent à reproduire un son périodique en synthétisant chacune des composantes sinusoïdales qui le constitue. Ces techniques, tout comme la synthèse par prédiction linéaire (LPC[196]) permettent d'obtenir des résultats de très grande qualité pour la re-synthèse et la modification de voix déjà enregistrées. Le principal inconvénient des techniques de synthèse de la voix chantée basées sur des modèles spectraux (à l'exception de la synthèse par modulation de fréquence) est qu'ils ne permettent pas de synthétiser des sons apériodiques et par conséquent les sons fricatifs de la voix comme les consonnes. Ainsi, dans la plupart des cas, il est nécessaire d'ajouter au modèle utilisé une source de bruit filtrée.

---

195 General Public Licence.

196 Linear Predictive Coding.

Les modèles basés sur la concaténation et la modification d'échantillons font partie des plus utilisés pour la synthèse de la voix parlée. Ils permettent d'obtenir des résultats de très grande qualité dans la mesure où ils sont basés sur des sons naturels. Ce type de modèle présente toutefois un certain nombre d'inconvénients. En effet, ils sont très gourmands en espace mémoire à cause de l'importance du nombre d'échantillons nécessaires et offrent une flexibilité et un contrôle très limité sur la synthèse.

Le domaine de la synthèse numérique de la voix chantée a fait des progrès significatifs au cours des vingt dernières années. Toutefois, bien que les résultats disponibles aujourd'hui soient intéressants, ils sont encore loin d'être comparables au son d'origine. Un nombre important de progrès restent par exemple à faire au niveau de l'expressivité et donc de l'intonation. Sur ce point, il serait intéressant d'améliorer l'automatisation de la production de paramètres pour la synthèse à partir d'une partition. Ceci impliquerait d'effectuer une analyse complète du comportement de chanteurs professionnels en fonction de différents contextes. Un tel travail se révèlerait toutefois être certainement colossal à cause du nombre de combinaisons possibles qui est pratiquement infini.

Chacune des techniques décrites précédemment présentant des qualités différentes, il pourrait être intéressant de les réunir au sein d'un même modèle comme le souligne Bruno Bossis :

> « Les barrières technologiques tombant les unes après les autres, des modèles aussi différents que ceux issus de la granulation, de la modulation de fréquence, de la prédiction linéaire et des fonctions d'onde formantique pourront être exploités au sein d'un même processus d'élaboration de la vocalité artificielle de façon à tirer parti des qualités de chacun.[197] »

L'amélioration de la synthèse des sons fricatifs produits par l'appareil phonatoire est aussi un domaine qui nécessite d'être étudié en particulier au niveau de l'enchainement avec des sons de type voisé.

Enfin, un travail important reste à effectuer au niveau de la création d'interfaces graphiques adaptées à l'analyse et à la synthèse des sons vocaux notamment au niveau de la représentation de certaines composantes du son : vibrato, etc.

L'Homme a toujours été fasciné par la voix chantée. Elle occupe un rôle important dans la plupart des civilisations dans la mesure où elle est un moyen de

---

197 BOSSIS, Bruno, *La voix et la machine : La vocalité artificielle dans la musique contemporaine*, Rennes : Presses Universitaire de Rennes, 2005, p. 290.

transmission musicale privilégiée. C'est très certainement le plus ancien des instruments de musique, mais aussi le plus complexe et le plus complet.

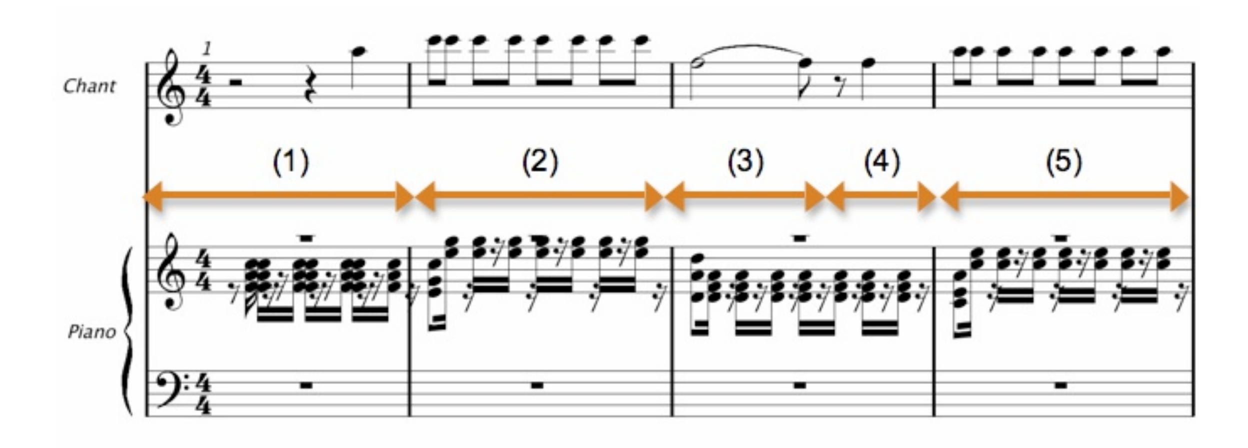
> « La voix s'est souvent révélée exemplaire des grandes articulations de l'histoire de la musique. De la naissance de la polyphonie avec l'école de Notre-Dame à l'émergence du romantisme dans les Lieder germaniques jusqu'au *Sprechgesang* d'Arnold Schönberg, la vocalité a non seulement toujours été présente, mais, par sa souplesse et son pouvoir expressif, elle a été un élément central dans de nombreuses expérimentations et remis en cause.[198] »

Au cours des derniers siècles, la fabrication de machines capables de chanter a été le but d'un grand nombre de scientifiques et d'inventeurs. Les résultats disponibles de nos jours paraitraient très certainement impressionnants pour nos ancêtres. Néanmoins, beaucoup de chemin reste encore à parcourir avant de voir un robot chanter au côté des plus grands interprètes sur une scène d'opéra.

---

198 BOSSIS, Bruno, *La voix et la machine : La vocalité artificielle dans la musique contemporaine*, Rennes : Presses Universitaire de Rennes, 2005, p. 283.

Annexes



Extrait de la partition issu du fichier Midi « reine-final.mid » pour la synthèse de *L'air de la reine de la nuit* de *La flute enchantée* de Mozart.

ANNEXE n° 1 : contenu du fichier MATLAB « fofd.m »

```
%Calcul d'une succession de Fonctions d'Onde Formantique selon le modèle de
CHANT

%Romain MICHON, Janvier 2010

freq=2000; %Fréquence de la FOF
Fs=44100; %Fréquence d'échantillonage
alpha=400;
time_sec=0.02; %Durée de la FOF
timek=time_sec.*Fs; %Nombre total d'échantillons
time=1./Fs:1./Fs:time_sec; %Vecteur temps pour le graph
beta=0.003; %ß en ms
beta2=pi./beta; %calcul de pi/ß
betak=beta.*Fs; %Conversion de ß en échantillons
x1=exp(-alpha.*time).*sin(2.*pi.*freq.*time);  %Calcul  des  pi/ß  premiers
échantillons
x2=((1-cos(beta2.*time)).*(exp(-alpha.*time).*sin(2.*pi.*freq.*time)))./2;
%Calcul des échantillons après pi/ß
x(1:betak)=x2(1:betak); %Sauvegarde du résultat final dans x
x((betak+1):timek)=x1((betak+1):timek);
lgtx=length(x); %Calcul la taille de x en nombre d'échantillons
t=0.005; %Période 1/FO
yt=t.*Fs; %Période en échantillons (correspond au début de la seconde FOF)
zt=t.*Fs.*2; %Début de la troisième FOF
y(1:yt)=0;
z(1:zt)=0;
y((yt+1):(lgtx+yt))=x; %Deuxième FOF
z((zt+1):(lgtx+zt))=x; %Troisième FOF
plot(x) %Affiche la première FOF
hold all
plot(y) %Affiche la seconde FOF
hold all
plot(z) %Affiche la troisième FOF
xlabel('echantillons k')
ylabel('amplitude de s')
```

ANNEXE n° 2 : contenu du fichier MATLAB « fof.m »

```
%Calcul d'une Fonction d'Onde Formantique selon le modèle de CHANT

%Romain MICHON, Janvier 2010

freq=2000; %Fréquence de la FOF
Fs=44100; %Fréquence d'échantillonage
alpha=400;
time_sec=0.02; %Durée de la FOF
timek=time_sec.*Fs; %Nombre total d'échantillons
time=1./Fs:1./Fs:time_sec; %Vecteur temps pour le graph
beta=0.003; %ß en ms
beta2=pi./beta; %calcul de pi/ß
betak=beta.*Fs; %Conversion de ß en échantillons
x1=exp(-alpha.*time).*sin(2.*pi.*freq.*time);  %Calcul  des  pi/ß  premiers
échantillons
x2=((1-cos(beta2.*time)).*(exp(-alpha.*time).*sin(2.*pi.*freq.*time)))./2;
%Calcul des échantillons après pi/ß
x(1:betak)=x2(1:betak); %Sauvegarde du résultat final dans x
x((betak+1):timek)=x1((betak+1):timek);
env1=exp(-alpha.*time); %Calcul de la forme de l'envellope
env2=((1-cos(beta2.*time)).*(exp(-alpha.*time)))./2;
env(1:betak)=env2(1:betak);
env((betak+1):timek)=env1((betak+1):timek);
%le fichier audio "fof-dem.wav" généré dans CSOUND est chargé
[y,Fsy]=wavread('/Users/romainmichon/Desktop/Memoire/Labete/Final/CD/csound
/fof-dem.wav');
subplot(2,1,1)
plot(x) %Affiche x en fonction du nombre d'échantillons
xlabel('echantillons k')
ylabel('amplitude de s')
title('FOF calculée d après les formules précédente')
%hold all
subplot(2,1,2)
plot(y)  %Affiche  le  contenu  de  "fof-dem.wav"  en  fonction  du  nombre
d'échantillons
%plot(env) %Affiche l'envellope en fonction du nombre d'échantillons
xlabel('echantillons k')
ylabel('amplitude de s')
title('FOF dans CSOUND')
```

ANNEXE n° 3 : contenu du fichier MATLAB « fofspec.m »

```
%Calcul et analyse FFT de 3 Fonctions d'Onde Formantique avec 3 valeurs
%de ß différentes

%Romain MICHON, Janvier 2010

freq=2000; %Fréquence des FOFs
Fs=44100; %Fréquence d'échantillonnage
alpha=80; %Alpha en Hz
time_sec=0.08; %Durée des FOFs
timek=time_sec.*Fs; %Nombre total d'échantillons
time=1./Fs:1./Fs:time_sec; %Vecteur 'time'
betaa1=0.01; %durée de l'attaque
betaa2=0.001;
betaa3=0.0001;
betab1=pi./betaa1; %calcul du paramètre ß
betab2=pi./betaa2;
betab3=pi./betaa3;
betak1=betaa1.*Fs; %Conversion des différentes valeurs de ß en échantillons
betak2=betaa2.*Fs;
betak3=betaa3.*Fs;
xa1=exp(-alpha.*time).*sin(2.*pi.*freq.*time);  %Calcul  des  pi/ß  premiers
échantillons
xb1=((1-cos(betab1.*time)).*(exp(-
alpha.*time).*sin(2.*pi.*freq.*time)))./2;  %Calcul  des  échantillons  après
pi/ß
x1(1:betak1)=xb1(1:betak1); %Sauvegarde du résultat final dans x1
x1((betak1+1):timek)=xa1((betak1+1):timek);
xa2=exp(-alpha.*time).*sin(2.*pi.*freq.*time);  %Calcul  des  pi/ß  premiers
échantillons
xb2=((1-cos(betab2.*time)).*(exp(-
alpha.*time).*sin(2.*pi.*freq.*time)))./2;  %Calcul  des  échantillons  après
pi/ß
x2(1:betak2)=xb2(1:betak2); %Sauvegarde du résultat final dans x2
x2((betak2+1):timek)=xa2((betak2+1):timek);
xa3=exp(-alpha.*time).*sin(2.*pi.*freq.*time);  %Calcul  des  pi/ß  premiers
échantillons
xb3=((1-cos(betab3.*time)).*(exp(-
alpha.*time).*sin(2.*pi.*freq.*time)))./2;  %Calcul  des  échantillons  après
pi/ß
x3(1:betak3)=xb3(1:betak3); %Sauvegarde du résultat final dans x3
x3((betak3+1):timek)=xa3((betak3+1):timek);

N=time_sec.*Fs; %Nombre total d'échantillon
faxis=(0:N-1).*Fs./N; %Vecteur 'fréquences' pour le graphs
fx1=pow2db(abs(fft(x1)./N))+11;   %Analyses   FFT   de   chaques   FOFs   avec
conversion de l'amplitude en db
fx2=pow2db(abs(fft(x2)./N))+11;
fx3=pow2db(abs(fft(x3)./N))+11;
plot(faxis,fx1) %Graph
xlabel('Fréquence (Hz)')
ylabel('Amplitude (db)')
hold all
plot(faxis(1:N-1),fx2)
hold all
plot(faxis(1:N-1),fx3)
```

ANNEXE n° 4 : contenu du fichier « voice-rules.Lisp » de la bibliothèque *PW-chant* de PatchWork

```
;;;;=========================================================
;;;;
;;;;  CHANT - PATCH-WORK
;;;;  By Mikael Laurson, Francisco Iovino.
;;;;  � 1991-1992 IRCAM
;;;;
;;;;=========================================================

(in-package :PW)

;==========================================================================
; RULE FOR THE AUTOMATIC BENDING OF THE FIRST TWO FORMANTS
;==========================================================================

(pw::add-menu-items *chant-voice-rule-menu* (pw::new-leafmenu "-" ()))

(defun calc-auto-bend (freq sex fund corr)
    (let ((f-elem (first freq)) (s-elem (second freq)))
        (when (<= f-elem fund)
            (incf f-elem (* (- fund f-elem) corr)))
        (unless (string= sex "ctn") ; castrato
           (when (and (>= s-elem 1300.) (>= fund 200.))
              (decf s-elem (* corr (* (* (/ 2 3) (- fund 200)) (/ (- s-elem
1300) 700)))))
           (when (<= s-elem (+ 30 (* 2. fund)))
              (incf s-elem (* corr (- (+ 30 (* 2 fund)) s-elem)))))
        (cons f-elem (cons s-elem (cddr freq)))))

(make-chant-vector-rule CHANT-USER::auto-bend
     ((fund chant-buffer)
      (corr chant-buff/fix/float (:value 1.0))
      (vfreq chant-vector))
      "auto-bend rule"
     (let ((fundamental-now (nth-rule-input self 1))
           (corr            (nth-rule-input self 2))
           (freq-list       (nth-rule-input self 3))
           (sex (formant-db-sex (formant-object (chant-synth self)))))
     (calc-auto-bend freq-list sex fundamental-now corr))
:menu *chant-voice-rule-menu*
)

;==========================================================================
; RULE FOR THE AUTOMATIC CALCULATION OF THE BANDWIDTHS
;==========================================================================
(defvar *chant-atb-pol-coefs* ())

(defun matrix-solver (matrix var)
  (let (u v w x y z d res1 res2 res3)
     (setq v (- (third (first matrix)) (third (second matrix))))
     (setq u (- (second (first matrix)) (second (third matrix))))
     (setq w (- (first var) (second var)))
     (setq x (- (second (first matrix)) (second (second matrix))))
     (setq y (- (second (second matrix)) (second (third matrix))))
     (setq z (/ x y))
     (setq d (- v (* (- (third (second matrix)) (third (third matrix)))
z)))
     (if (and (not (zerop x)) (not (zerop u)) (not (zerop d)))
       (progn (setq res3 (/ (* (- w (- (second var) (third var))) z) d))
              (setq res2 (/ (- w (* res3 v)) x))
```

```
              (setq res1 (- (first var) (+ (* res2 (second (first matrix)))
(* res3 (third (first matrix))))))
              (list res1 res2 res3))
       (print "ERROR IN MATRIX RESOLUTION."))))

(defclass C-atb-class ()
  ((chant-ref-freq  :initform '(200 500 4000) :accessor chant-ref-freq)
   (chant-ref-bw    :initform '(75  75  150) :accessor chant-ref-bw)))

(defmethod get-pol-coefs ((self C-atb-class))
   (let (matrix temp)
    (list 1 (log (first (chant-ref-freq self))) (* (log (first (chant-ref-
freq self))) (log (first (chant-ref-freq self)))))
    (for (i 0 1 2)
        (setq temp (log (nth i (chant-ref-freq self))))
        (push (list 1 temp (* temp temp)) matrix))
    (setq matrix (nreverse matrix))
    (setq   *chant-atb-pol-coefs*   (matrix-solver   matrix   (chant-ref-bw
self)))))

(defmethod calc-auto-band ((self C-atb-class) freq)
   (let (result temp (nof (length freq)))
      (for (i 0 1 (1- nof))
          (setq temp (log (nth i freq)))
          (push (+ (+ (first *chant-atb-pol-coefs*) (* temp (second *chant-
atb-pol-coefs*)))
                        (* (* temp temp) (third *chant-atb-pol-coefs*)))
                 result))
       (nreverse result)))

(defvar *atb-rule-object* (make-instance 'C-atb-class))
(get-pol-coefs *atb-rule-object*)
;(calc-auto-band *atb-rule-object* '(609 1000 2450 2700 3240))

(make-chant-vector-rule CHANT-USER::auto-bw
   ((vfreq chant-vector))
    "auto-bend rule"
    (let ((freq-list (nth-rule-input self 1)))
      (calc-auto-band *atb-rule-object* freq-list))
:menu *chant-voice-rule-menu*
)

;===========================================================================
; RULE FOR THE AUTOMATIC CALCULATION OF THE AMPLITUDES
;===========================================================================

(defvar *chant-atb-pol-coefs* ())

(defclass C-ata-class ()
  ((chant-chp  :initform (make-array 12) :accessor chant-chp)))

(defmethod initialize-instance :after ((self C-ata-class) &rest args)
   (declare (ignore args))
   (setf (svref (chant-chp self) 0) 1.0)
   (setf (svref (chant-chp self) 1) (expt 10 (/ 1 20)))
   (setf (svref (chant-chp self) 2) (expt 10 (/ 2.5 20)))
   (setf (svref (chant-chp self) 3) (expt 10 (/ 5 20)))
   (setf (svref (chant-chp self) 4) (expt 10 (/ 9 20)))
   (setf (svref (chant-chp self) 5) (expt 10 (/ 14.5 20)))
   (setf (svref (chant-chp self) 6) (expt 10 (/ 21.5 20)))
   (setf (svref (chant-chp self) 7) (expt 10 (/ 30. 20)))
   (setf (svref (chant-chp self) 8) (expt 10 (/ 41. 20)))
   (setf (svref (chant-chp self) 9) (expt 10 (/ 51. 20)))
   (setf (svref (chant-chp self) 10) (expt 10 (/ 55. 20)))
```

```
   (setf (svref (chant-chp self) 11) (expt 10 (/ 55. 20))))

(defvar *ata-rule-object* (make-instance 'C-ata-class))

(defun get-chant-chp-index (f)
   (let ((i (truncate (/ f 500))))
      (cond ((>= i 11) 11)
            ((<= i 0) 0)
            (t i))))

;(get-chant-chp-index 1200)

(defun get-chant-cont-factor (cf f-list bw-list)
    (let ((result 1.) mbw mbw2 numer denom fi)
       (for (i 0 1 (1- (length f-list)))
          (setq mbw (max (nth i bw-list) 0.000001))
          (setq mbw2 (/ (* mbw mbw) 4))
          (setq fi (nth i f-list))
          (setq numer (+ (* fi fi) mbw2))
          (setq denom (* (expt (+ mbw2 (* (- fi cf) (- fi cf))) 0.5)
                         (expt (+ mbw2 (* (+ fi cf) (+ fi cf))) 0.5)))
          (setq result (* result (/ numer denom))))
       result))

(defmethod calc-auto-ampl ((self C-ata-class) freq bw)
   (let (result ind  chp-cont fi)
      (for (i 0 1 (1- (length freq)))
         (setq fi (nth i freq))
         (setq ind (get-chant-chp-index fi))
         (setq chp-cont (+ (svref (chant-chp self) ind)
                           (*  (-  (svref  (chant-chp  self)  (1+  ind))  (svref
(chant-chp self) ind))
                              (/ (- fi (* 500 ind)) 500))))
         (push (* (get-chant-cont-factor fi freq bw)
                  (/ chp-cont fi)) result))
         (nreverse result)))

(defmethod calc-auto-ampl ((self C-ata-class) freq bw)
   (let (result ind  chp-cont fi)
      (for (i 0 1 (1- (length freq)))
         (setq fi (nth i freq))
         (setq ind (get-chant-chp-index fi))
         (setq chp-cont (+ (svref (chant-chp self) ind)
                           (* (- (svref (chant-chp self)
                                   (if (= ind 11) ind (1+ ind)))
                   ;;;Array index 12 out of bounds for #<SIMPLE-VECTOR 12>
                                   (svref (chant-chp self) ind))
                              (/ (- fi (* 500 ind)) 500))))
         (push (* (get-chant-cont-factor fi freq bw)
                  (/ chp-cont fi)) result))
         (nreverse result)))

(make-chant-vector-rule CHANT-USER::auto-ampl
  ((vfreq chant-vector)
   (vbw chant-vector))
    "auto-ampl rule"
   (let ((freq-list (nth-rule-input self 1))
         (bw-list (nth-rule-input self 2)))
      (calc-auto-ampl *ata-rule-object* freq-list bw-list))
:menu *chant-voice-rule-menu*
)
```

```
;============================================================================
; RULE FOR THE VOCAL EFFORT CORRECTION
;============================================================================

(defun ampl-correction (cslope fund f-mean ajus sex amp coefamp envelo freq
ampl)
   (let (result tino rslope amp-i)
      (when (zerop cslope) (setq cslope 0.000001))
      (setq tino (* coefamp (expt (/ fund f-mean) (third ajus))))
      (setq rslope (* cslope (exp (* (first ajus)
                              (atan (* (second ajus) (log (/ fund f-
mean))))))))
      (do ((i 0 (+ 1 i))) ((eq i (length ampl)))
        (setq amp-i (* envelo tino (nth i ampl)))
        (if (> (nth i freq) (first freq))
          (if (< cslope 0)
            (if (or (string= sex "ctn") (string= sex "bas")(string= sex
"ten")(string= sex "mal"))
              (progn
;                (print (list "foo" i tino amp-i))
                (push (* amp-i amp coefamp (+ 3 (/ (- 400 fund) 300)
                                     (* .1 (/ (- 400 fund) 300))))
                      result))
              (push (* amp-i amp coefamp (+ .8 (/ (- 1000 fund) 1250)
                                     .05 (* .1 (/ (- 1000 fund)
1250))))
                    result))
            (push (* amp-i rslope) result))
          (push amp-i result)))
         (nreverse result)))

(make-chant-vector-rule CHANT-USER::ampl-corr
  ((envelo chant-buffer)
   (vfreq chant-vector)
   (vamp chant-vector))
      "amplitude-correction rule"
  (let ((envelo     (nth-rule-input self 1))
        (freq-list  (nth-rule-input self 2))
        (ampl-list  (nth-rule-input self 3))
        (sex (formant-db-sex (formant-object (chant-synth self)))))
      (ampl-correction -1 100 200 '(0. 5.7 0.) sex 1. 1. envelo freq-list
ampl-list))
:menu *chant-voice-rule-menu*
)
```

ANNEXE n° 5 : contenu du fichier CSOUND « fof-sop.csd »

```
;Synthèse d'une voix de soprano chantant la voyelle 'a' avec interpolation
entre chaque note.

;Romain MICHON, Janvier 2010

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

;Variables permettant de mémoriser la fréquence de la note jouée
précédemment, elles doivent êtres initialisées à la fréquence de la
première note jouée
gilasts init 440
gilasta init 440

instr 1 ;Soprano chantant un 'a'

kris init .003 ;Durée de l'attaque des impulsions
kdur init .02 ;Durée totale des impulsions
kdec init .007 ;Durée du decay des FOF
iolaps = 15
ifna = 1 ;Utilisation d'une table de fonction de sinusoïde (GEN10) pour les
impulsions
ifnb = 2 ;Utilisation d'une table de fonction de sigmoïde (GEN19) pour la
création de l'enveloppe
itodur = 440 ;fréquence de production des impulsions
iskip = 1 ;supprime l'attaque de chaque objet 'fof' autorisant ainsi les
legato

ifre1 = 800 ;fréquences des formants d'après les valeurs données aux pages
2350-2354 du manuel de Csound
ifre2 = 1150
ifre3 = 2900
ifre4 = 3900
ifre5 = 4950

iamp1 = p5 ;amplitudes des formants d'après les valeurs données aux pages
2350-2354 du manuel de Csound
iamp2 = iamp1 - 6
iamp3 = iamp1 - 32
iamp4 = iamp1 - 20
iamp5 = iamp1 - 50

iwid1 = 80 ;largeurs de bandes des formants d'après les valeurs données aux
pages 2350-2354 du manuel de Csound
iwid2 = 90
iwid3 = 120
iwid4 = 130
iwid5 = 140

aint linseg gilasts, .05, p4 ;Permet l'interpolation des différents
formants d'une note à l'autre
gilasts = p4 ;enregistre la valeur de la fréquence actuelle pour la
réutiliser lors de la prochaine interpolation

amod oscil 20, 5, 1 ;signal modulant la fréquence de chaque formant
(vibrato)
```

```
;amod = 0 ;si amod = 0, le vibrato est désactivé

;Creation des 5 formants
ar1 fof ampdb(iamp1), aint+amod, ifre1, 0, iwid1, kris, kdur, kdec, iolaps,
ifna, ifnb, itodur, 0, 0, iskip
ar2 fof ampdb(iamp2), aint+amod, ifre2, 0, iwid2, kris, kdur, kdec, iolaps,
ifna, ifnb, itodur, 0, 0, iskip
ar3 fof ampdb(iamp3), aint+amod, ifre3, 0, iwid3, kris, kdur, kdec, iolaps,
ifna, ifnb, itodur, 0, 0, iskip
ar4 fof ampdb(iamp4), aint+amod, ifre4, 0, iwid4, kris, kdur, kdec, iolaps,
ifna, ifnb, itodur, 0, 0, iskip
ar5 fof ampdb(iamp5), aint+amod, ifre5, 0, iwid5, kris, kdur, kdec, iolaps,
ifna, ifnb, itodur, 0, 0, iskip

asort = ar1 + ar2 + ar3 + ar4 + ar5 ;addition des 5 formants
out asort

endin
```

ANNEXE n° 6 : contenu du fichier MATLAB « fof.m »

```
%Script affichant la largeur de bande (bw) d'une Fonction d'Onde
%Formantique en fonction de sa fréquence selon la formule appliquée par
%CHANT

%Romain MICHON, Janvier 2010

freq=[0:1:2000]; %Création d'un vecteur 'fréquence' pour le graph
bw = 471 - 138.6 .* reallog(freq) + 12.04 .* (reallog(freq)).^2; %calcul de
la largeur de bande
plot(bw) %Création d'un graphique
xlabel('fréquence (Hz)')
ylabel('largeur de bande (Hz)'
```

ANNEXE n°7 : fonctionnement de la fonction *exec-csound* de la bibliothèque *chant-lib*

Dans un premier temps, le nom de la fonction est défini ainsi que ses différents arguments : la liste envoyée par la fonction *chant* à convertir en partition (*resform*) et le nom du fichier audio produit (*out-name*).

```
(om::defmethod! exec-csound (resformf out-name)
```

Création de la partition CSOUND :

Les variables locales sont déclarées.

```
(let* ((normalize) (resolution)
```

filename contient le nom et le chemin d'accès de la future partition « my-sco.sco ». Elle sera automatiquement placée dans le dossier « out-files » du workspace.

```
   (filename (make-pathname
             :directory (pathname-directory
             *om-tmpfiles-folder*)
     :name "my-sco"
     :type "sco")))
```

La partition CSOUND est créée à partir de la variable *resformf* et est placée dans la variable *lst?*.

```
 (setf  lst? (list! (reverse resformf)))
 (setf  lst? (mapcar #'(lambda (x) (if (typep x 'son-obj)
                                     (insts->sco x) (list x))) lst?))
 (setf lst? (om::flat-once (mapcar #'flat-max-1 (remove '() lst?))))
 (setf *lastcsdfile* filename)
```

La fonction *with-open-file* crée un fichier dont le nom et le chemin d'accès sont définis par *filename* et le contenu par *fd*. Si un fichier du même nom existe déjà dans le répertoire spécifié, il est remplacé.

```
 (with-open-file  (fd filename :direction :output :if-exists :rename-and-
delete
                              :if-does-not-exist :create)
 (let ((glou nil))
```

Les différents événements CSOUND ayant été placés dans des listes auparavant, le nombre de listes est détecté et les opérations suivantes sont répétées en fonction du nombre d'événements.

```
 (dotimes (n (length lst?))
```

Les lignes de commentaires sont générées avec la fonction *print-comment* de *om2csound* et sont placées dans la variable *glou*. Elles contiennent le numéro de chaque p-field de chaque événement CSOUND.

```
    (setf glou (print-comment (nth n lst?) fd glou))
```

La fonction *printscoseq* de *om2csound* lit chacune des listes contenues dans la variable *lst?*, les transforme en événements CSOUND et place le résultat dans *fd*. La partition CSOUND se

termine par la lettre « e » indiquant au compilateur l'emplacement de la fin de l'instrument. Enfin, l'utilisateur est averti dans le *listener* du nom et de l'emplacement du fichier partition créé.

```
        (printscoseq  (nth n lst?) fd)
        (format fd "~%"))
    (format fd "e")))
  (print (string+ "CREATION D'UN FICHIER SCO A L'ADRESSE SUIVANTE :"
  (namestring filename)))
```

Synthèse avec CSOUND :

Création d'une variable globale contenant les flags par défauts de CSOUND pour la compilation.

```
  (setf *flags* *csound-defflags*)
```

On vérifie si CSOUND est installé sur la machine. Si c'est le cas, les opérations suivantes sont exécutées sinon un message d'erreur est affiché dans le listener : « IMPOSSIBLE DE TROUVER CSOUND ».

```
    (if (probe-file *CSOUND-PATH*)
```

Une normalisation étant appliquée au fichier audio créé, il sera d'abord placé dans un fichier temporaire : « norm.tmp ».

Le nom et le chemin d'accès du fichier audio final sont sauvegardés dans la variable *outpath*. *om-outfiles-folder* indique que ce dernier sera placé dans le dossier « out-files » du workspace.

```
        (let* ((tmppath (tmpfile "norm.tmp"))
            (outpath (corrige-sound-filename (if out-name out-name "snd-
out")
            *om-outfiles-folder*)))
```

On vérifie d'abord si un fichier porte le même nom que celui défini lors de l'étape précédente dans le dossier « out-files » du worspace. Si c'est le cas, il est effacé et un message s'affiche dans le listener.

```
        (when (probe-file outpath)
          (print (string+ "SUPRESSION DU FICHIER EXISTANT : "
          (namestring outpath)))
          (om-delete-file outpath))
```

Le compilateur de CSOUND est appelé directement dans le terminal grâce à la fonction *om-cmd-line*. Cette dernière permet d'envoyer dans le terminal une commande contenue dans une chaîne de caractères. Dans le cas présent, la commande est constituée :

- du chemin d'accès à CSOUND contenu dans *CSOUND-PATH*
- des flags définissants les paramètres de la compilation : *flags*
- du nom et du chemin d'accès de l'orchestre CSOUND : « chant.orc » qui doit être placé au préalable dans le dossier « in-files » du workspace

- du nom et du chemin d'accès de la partition CSOUND contenue dans la variable *filename*
- du nom et de l'emplacement du fichier audio à créer, en l'occurrence, ceux du fichier temporaires « norm.tmp » contenus dans *tmppath*.

```
        (print "SYNTHESE AVEC CSOUND...")
        (om-cmd-line (print (format nil "~s ~A ~s ~s -o ~s"
            (om-path2cmdpath *CSOUND-PATH*)
                *flags*
                (om-path2cmdpath (make-pathname
                :directory (pathname-directory *om-infiles-folder*)
                        :host (pathname-host *om-infiles-folder*)
                        :name (pathname-name "chant.orc")
                        :type (pathname-type "chant.orc")))
                    ;affiche et deffini l'emplacement de la partition pour
CSOUND
                (om-path2cmdpath filename)
                (om-path2cmdpath tmppath)))
         *sys-console*)
```

Comme cela a été indiqué précédemment, le fichier audio temporaire créé (norm.tmp) est normalisé à zéro décibels afin d'homogénéiser le son. Ceci est réalisé avec la fonction *general-normalize*. Le fichier audio produit lors de cette opération sera placé à l'emplacement spécifié par la variable *outpath*.

Une fois la normalisation effectuée, les fichiers temporaires sont effacés et un message indiquant le nom et l'emplacement du fichier final est envoyé au listener.

```
            (let ((real-out (general-normalize *normalizer*
               tmppath outpath normalize resolution)))
         (push tmppath *tmpparfiles*))
         (when *delete-inter-file* (clean-tmp-files))
        (probe-file outpath))
(om-beep-msg "CSOUND NOT FOUND"))))
```

ANNEXE n°8 : explication du fonctionnement de la fonction *chant* de la bibliothèque *chant-lib*

Dans un premier temps, le nom de la fonction est défini ainsi que ses différents arguments. Des valeurs initiales sont données à chaque entrée afin que la synthèse puisse s'effectuer même si l'une des entrées de la fonction n'est pas renseignée.

```
(om::defmethod! chant ((onset  t) (duree t) (f0 t) (envelope list)
   (vibra list) (ampf list) (ffreq list)
   (bdwth list) &optional (out-name t))

  :icon 150
  :indoc '("onset" "durees" "f0" "enveloppe" "vibrato" "jitter"
     "amplitudes des formants" "frequences de formants"
     "largeurs de bande des formants" "Nom du fichier audio")
  :initvals '(0 1 440 '(0.1 0.1 0.15 '(70))
  '(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
  '((0)) '((0)) '((0)) '"chant.aif")
```

Le nombre de notes à jouer est calculé à partir de la liste *onset* et est sauvegardé dans la variable *notes*. Les opérations suivantes seront répétées en fonction de la valeur de *notes* afin de créer une liste pour chaque note.

```
(if (and (= (length duree) (length onset)))
  (let* ( (notes (length onset)) (resform) (resformf))
    (dotimes (n notes resform)
```

L'étape suivante permet la détection des fonctions sur les différentes entrées. Comme cela a été dit précédemment, le code de cette section sera présenté ultérieurement au cas par cas lors de l'étude des autres fonctions afin de faciliter la compréhension.

```
[…]
```

Une liste *resform* est créée avec l'ensemble des listes de paramètres fournis à la fonction *chant* pour chaque note. Chaque élément d'une liste correspond alors à un p-field d'événement CSOUND. *resform* est donc de la forme ((*évènement 1*) (*événement 2*) (*événement n*)).

resform sera traitée par la fonction *exec-csound* qui la transformera en partition et l'exécutera dans CSOUND avec l'orchestre « chant.orc ». Les premiers éléments de chaque liste sont « i » pour *instrument* dans CSOUND et « 1 » pour *instrument n°1*.

```
(push   (list 'i '1
```

Les valeurs des onsets, des durées, des fréquences et des amplitudes sont extraites pour chaque note.

```
(nth n onset) (nth n duree)
(if (= (length f0) '1) (nth 0 f0) (nth n f0))
(if (= (length amps) '1) (nth 0 amps) (nth n amps))
```

Les paramètres du vibrato et du jitter sont définis. Dans l'ordre : *vibamp*, *vala1*, *vala2*, *tvala1*, *tvala2*, *vibfreq*, *valf1*, *valf2*, *tvalf1*, *tvalf2*, *jitt1*, *jitt2*, *jitt3*, *jittf1*, *jittf2*, *jittf3*. Ces paramètres n'étant pas dynamiques, leurs valeurs resteront les même pour chaque notes.

```
(nth 0 vibra) (nth 1 vibra) (nth 2 vibra) (nth 3 vibra)
(nth 4 vibra) (nth 5 vibra) (nth 6 vibra) (nth 7 vibra)
(nth 8 vibra) (nth 9 vibra) (nth 10 vibra) (nth 11 vibra)
(nth 12 vibra) (nth 13 vibra) (nth 14 vibra) (nth 15 vibra)
```

Les paramètres relatifs à la fonction *envelope* sont insérés dans *resform*. Leur valeur sont statique et restent les mêmes pour chaque note (cf. (III)-(B)-(f)).

```
att rel interp notes
```

Les valeurs des amplitudes, fréquences et largeurs de bandes des formants sont ajoutées à *resform*. Ces valeurs sont dynamiques.

```
amp1 amp2 amp3 amp4
freq1 freq2 freq3 freq4 freq5
bdw1 bdw2 bdw3 bdw4 bdw5
```

Le traitement des informations pour la correction de l'amplitude des formants se faisant directement dans l'orchestre CSOUND, il est nécessaire d'envoyer les paramètres à cette règle (cf. (III)-(B)-(i)). Ceci est du au caractère évolutif de l'amplitude d'une note au cours du temps (enveloppe). Les informations doivent donc être traitées en direct.

```
cslope ajus1 ajus2
ajus3 f-mean sexef
```

Le paramètre suivant permet d'indiquer à l'orchestre CSOUND laquelle des enveloppes utiliser : *envelope* ou *envelope2*.

```
envgen)
resform))
```

Trois tables de fonction sont utilisées par l'orchestre CSOUND « chant.orc » :

- Une créant une sinusoïde (GEN10) pour les impulsions des générateurs de FOFs (cf. (II)-(A)-(c)).
- Une créant une sigmoïde (GEN19) pour définir l'enveloppe de chaque impulsion (cf. (II)-(A)-(c)).
- La dernière table de fonction est optionnelle, elle est ici sauvegardée dans la variable *fenv*. Ses paramètres sont générés par *envelope2* en fonction de la forme dessinée dans le BPF qui lui est associé (cf. (III)-(B)-(f)). Elle permet donc de décrire la forme de l'enveloppe d'amplitude des différentes notes produites.

Ces tables de fonctions sont sauvegardées sous forme de liste au début de *resform*.

```
(push (append
(list '(f 1 0 8192 10 1))
(list '(f 2 0 1024 19 0.5 0.5 270 0.5))
(list fenv)
(reverse resform)
)
resformf)
```

La dernière opération effectuée par la fonction *chant* consiste à envoyer la liste de paramètres créée *resformf* à la fonction *exec-csound* qui la transformera en partition CSOUND puis l'exécutera avec l'orchestre « chant.orc ».

```
(exec-csound resformf out-name)))
```

ANNEXE n° 9 : contenu du fichier CSOUND « chant.orc »

```
;Cet Orchestre CSOUND est rattaché à la bibliothèque "synth-chant"
d'OpenMusic.
;Il implémente toutes les tâches de traitement du signal demandé par la
méthode "chant".
;Cette dernière créée une partition CSOUND à partir d'une liste de
paramètres, cette partition est ensuite
;appliqué à l'orchestre suivant.

;Romain MICHON, mai 2010, "La synthèse de la voix chantée par Fonctions
d'Onde Formantique -
;techniques, outils existants, exemple d'implémentation et d'utilisation",
master de musicologie.

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

gilast init 0 ;déclaration des variables globales
gifre1 init 0
gifre2 init 0
gifre3 init 0
gifre4 init 0
gifre5 init 0
giwid1 init 0
giwid2 init 0
giwid3 init 0
giwid4 init 0
giwid5 init 0
giamp1 init 0
giamp2 init 0
giamp3 init 0
giamp4 init 0
giamp5 init 0
gilast_tst init 0
gicnt init 0

instr 1

koct = 0
kris = .003 ;Durée de l'attaque des impulsions
kdur = .02 ;Durée totale des impulsions
kdec = .007 ;Durée du decay des FOF
iolaps = 100000
ifna = 1 ;Utilisation d'une table de fonction de sinusoïde (GEN10) pour les
impulsions
ifnb = 2 ;Utilisation d'une table de fonction de sigmoïde (GEN19) pour la
création de l'enveloppe
itotdur = p3 ;durée totale de la note

iphs = 0 ;phase initiale de la fondamentale
ifmode = 1 ;permet d'utiliser des fréquences de formants différentes pour
chaque notes
iinterp = p24
kenvar1 = 1 ;si la règle amp-cor n'est pas utilisée, kenvar1 doit être
égale à 1
aengar1 = 1 ;si la règle amp-cor n'est pas utilisée, aengar1 doit être
égale à 1

if0 = p4 ;fréquence de la fondamentale
```

```
;si la méthode "ffreq" n'est pas connecté à la méthode "chant" dans OM,
l'orchestre CSOUND utilise des valeurs par défaut
if (p30==0 && p31==0 && p32==0 && p33==0 && p34==0) then

      ifre1 = 800 ;fréquences par défaut des formants
      ifre2 = 1150
      ifre3 = 2900
      ifre4 = 3900
      ifre5 = 4950
else
      ifre1 = p30 ;fréquences des formants définies dans OM
      ifre2 = p31
      ifre3 = p32
      ifre4 = p33
      ifre5 = p34
endif

;si la méthode "ampf" n'est pas connecté à la méthode "chant" dans OM,
l'orchestre CSOUND utilise des valeurs par déffaut
if (p26 == 0 && p27 == 0 && p28 == 0 && p29 == 0) then

      ;amplitudes par déffaut des formants des formants traduites
      de dB à 0dbfs = 32767

      iamp1 = ampdb(p5)
      iamp2 = ampdb(p5 - 6)
      iamp3 = ampdb(p5 - 32)
      iamp4 = ampdb(p5 - 20)
      iamp5 = ampdb(p5 - 50)
else
      ;amplitudes des formants déffinies dans OM traduites de db à 0dbfs =
      32767

      iamp1 = ampdb(p5)
      iamp2 = ampdb(p5 + p26)
      iamp3 = ampdb(p5 + p27)
      iamp4 = ampdb(p5 + p28)
      iamp5 = ampdb(p5 + p29)

endif

;si la méthode "bdwth" n'est pas connecté à la méthode "chant" dans OM,
l'orchestre CSOUND utilise des valeurs par déffaut
if (p35 == 0 && p36 == 0 && p37 == 0 && p38 == 0 && p39 == 0) then
      iwid1 = 80 ;largeurs de bandes par déffaut des formants
      iwid2 = 90
      iwid3 = 120
      iwid4 = 130
      iwid5 = 140
else
      iwid1 = p35 ;largeurs de bandes déffinies dans OM
      iwid2 = p36
      iwid3 = p37
      iwid4 = p38
      iwid5 = p39
endif


;si la méthode "vibra" n'est pas connectée à la méthode "chant", pas de
vibrato
if (p6=0 && p7=0 && p8=0 && p9=0 && p10=0 && p11=0 && p12=0 && p13=0 &&
p14=0 && p15=0) then
      amod = 0
```

```
else
	;modulation de l'amplitude du vibrato
	kvala1 = p7
	ktvala1 = p9
	kvala2 = p8
	ktvala2 = p10

	;des valeurs sont tirées au sort dans une gamme définie par kvala1 à
	une fréquence de ktvala1
	krand_amp1 randh kvala1*1000, ktvala1
	kavar_amp1 = 1+((krand_amp1/1000) - (kvala1/2))
	;chaque valeur est interpolée avec la précédente et la suivante
	kavar_amp_interp1 lineto kavar_amp1, 1/ktvala1

	krand_amp2 randh kvala2*1000, ktvala2
	kavar_amp2 = 1+((krand_amp2/1000) - (kvala2/2))
	kavar_amp_interp2 lineto kavar_amp2, 1/ktvala2

	;modulation de la fréquence du vibrato
	kvalf1 = p12
	ktvalf1 = p14
	kvalf2 = p13
	ktvalf2 = p15

	;des valeurs sont tirées au sort dans une gamme définie par kvalf1 à
	une fréquence de ktvalf1
	krand_freq1 randh kvalf1*1000, ktvalf1
	kavar_freq1 = 1+((krand_freq1/1000) - (kvalf1/2))
	kavar_freq_interp1 lineto kavar_freq1, 1/ktvalf1

	krand_freq2 randh kvalf2*1000, ktvalf2
	kavar_freq2 = 1+((krand_freq2/1000) - (kvalf2/2))
	kavar_freq_interp2 lineto kavar_freq2, 1/ktvalf2

	;paramètres de l'enveloppe du vibrato
	iatt_vib = p47
	idurn_vib = p3-0.02
	idec_vib = p48

	;si la fonction envelope-vib est connectée, alors une enveloppe est
	appliquée au vibrato
	if (iatt_vib != 0 && idec_vib != 0) then
		kenv_vib linen 1, iatt_vib, idurn_vib, idec_vib
	else
		kenv_vib = 1
	endif

	;création du vibrato
	kvibamp = p6
	kvibfreq = p11
	;signal modulant pour la fondamentale
	amod oscili kvibamp*((kavar_amp_interp1+kavar_amp_interp2)/2),
	kvibfreq*((kavar_freq_interp1+kavar_freq_interp2)/2), 1
endif


;jitters pour les variations aléatoires de la fondamentale

;la méthode "jitter" est-elle connectée à la méthode "chant"?
if (p16=0 && p17=0 && p18=0 && p19=0 && p20=0 && p21=0) then
	kfond_var = 0
else
	kjitt1 = p16
	kjitt2 = p17
```

```
      kjitt3 = p18
      kjittf1 = p19
      kjittf2 = p20
      kjittf3 = p21

      kfond_var1 jitter kjitt1*if0, kjittf1, kjittf1
      kfond_var2 jitter kjitt2*if0, kjittf2, kjittf2
      kfond_var3 jitter kjitt3*if0, kjittf3, kjittf3
      ;le jitter final est composé d'une moyenne des trois précédents
      kfond_var = (kfond_var1 + kfond_var2 + kfond_var3)/3
endif

;Interpolation des fréquences, bandwidths et amplitudes des formants d'une
note à une autre
if (iinterp==0) then
      iskip = 0
else
      iskip = 1
endif

;pour empêcher l'interpolation si elle est désactivée ou lors de la
première note
if (gilast_tst>0 && iinterp!=0) then
      iduri = iinterp ;durée de l'interpolation
      af0i linseg gilast, iduri, p4 ;interpolation linéaire de la
fondamentale

      ;interpolation linéaire de la fréquence de chaque formant
      afre1i linseg gifre1, iduri, ifre1
      afre2i linseg gifre2, iduri, ifre2
      afre3i linseg gifre3, iduri, ifre3
      afre4i linseg gifre4, iduri, ifre4
      afre5i linseg gifre5, iduri, ifre5

      ;interpolation linéaire de la largeur de bande de chaque formant
      kwid1i linseg giwid1, iduri, iwid1
      kwid2i linseg giwid2, iduri, iwid2
      kwid3i linseg giwid3, iduri, iwid3
      kwid4i linseg giwid4, iduri, iwid4
      kwid5i linseg giwid5, iduri, iwid5

      ;interpolation linéaire des amplitudes des formants
      aamp1i linseg giamp1, iduri, iamp1
      aamp2i linseg giamp2, iduri, iamp2
      aamp3i linseg giamp3, iduri, iamp3
      aamp4i linseg giamp4, iduri, iamp4
      aamp5i linseg giamp5, iduri, iamp5

      ;Variables globales sauveguardant les paramètres des notes
      précédentes pour l'interpolation
      gilast = p4
      gifre1 = ifre1 ;Variables globales pour les fréquences des formants
      gifre2 = ifre2
      gifre3 = ifre3
      gifre4 = ifre4
      gifre5 = ifre5

      giwid1 = iwid1 ;Variables globales pour les largeurs de bandes
      giwid2 = iwid2
      giwid3 = iwid3
      giwid4 = iwid4
      giwid5 = iwid5

      giamp1 = iamp1
```

```
      giamp2 = iamp2 ;Variables globales pour les amplitudes des formants
      giamp3 = iamp3
      giamp4 = iamp4
      giamp5 = iamp5
;pas d'interpolation lors de la première note ou si elle est désactivée
else
      gilast = p4

      gifre1 = ifre1
      gifre2 = ifre2
      gifre3 = ifre3
      gifre4 = ifre4
      gifre5 = ifre5

      giwid1 = iwid1
      giwid2 = iwid2
      giwid3 = iwid3
      giwid4 = iwid4
      giwid5 = iwid5

      giamp1 = iamp1
      giamp2 = iamp2
      giamp3 = iamp3
      giamp4 = iamp4
      giamp5 = iamp5

      af0i = p4

      afre1i = ifre1
      afre2i = ifre2
      afre3i = ifre3
      afre4i = ifre4
      afre5i = ifre5

      kwid1i = iwid1
      kwid2i = iwid2
      kwid3i = iwid3
      kwid4i = iwid4
      kwid5i = iwid5

      aamp1i = iamp1
      aamp2i = iamp2
      aamp3i = iamp3
      aamp4i = iamp4
      aamp5i = iamp5
      gilast_tst = 1
endif

;paramètres fournis par la fonction "envelope" utilisés soit pour créer une
enveloppe globale sur la phrase musicale entière, soit pour une enveloppe
locale sur chaque notes
idurn = p3-0.02
iatt = p22
idec = p23

;si l'interpolation entre les fréquences de la fondamentale est desactivée,
les paramètres de l'objet "envelope" sont appliqués à chaque note
if (iinterp=0) then
      ;contrôle de l'enveloppe globale des notes
      aenvg = 1 ;pas d'enveloppe globale de la phrase musicale
      ;p46 est un p-field test généré par la méthode "chant"
      if (p46=1) then
            ;utilisation des paramètres de la méthode "enveloppe2"
            kenv oscil1 0, 1, idurn, 3
```

```
        else
              ;utilisation des paramètres de la méthode "envelope"
              kenv linen 1, iatt, idurn, idec
        endif
else
        ;contrôle de l'enveloppe globale de la phrase musicale :
        ;les paramètres de la méthode "envelope" sont utilisés
        ;si la fonction envelope-vib est connectée, alors une enveloppe
        ;globale est appliquée au vibrato
        knotes = p25
        kenv = 1
        if (gicnt == 0) then
              aenvg linseg 0, iatt, 1, (idurn - iatt), 1
        elseif (gicnt == (knotes -1)) then
              aenvg linseg 1, (idurn - idec), 1, idec, 0
        else
              aenvg = 1
        endif
        gicnt = gicnt + 1

endif

;correction automatique de l'amplitude des formants : méthode "amp-cor"
if (p40==0 && p41==0 && p42==0 && p43==0 && p44==0 && p45==0) then ;la
méthode "amp-cor" est-elle connectée?
else

        kcslope = p40 ;pente du spectre
        kajus1 = p41 ;coefficient d'ajustement n°1
        kajus2 = p42 ;coefficient d'ajustement n°2
        kajus3 = p43 ;coefficient d'ajustement n°3
        kfmean = p44 ;fréquence médiane de la tessiture de la voix utilisée
        ksexef = p45 ;male ou femelle?

              ;calcul
              atino = kenv*aenvg*(if0/kfmean)^kajus3
              krslope = kcslope*exp(kajus1*taninv(kajus2*log(if0/kfmean)))
              aamptmp2 = atino*aamp2i
              aamptmp3 = atino*aamp3i
              aamptmp4 = atino*aamp4i
              aamptmp5 = atino*aamp5i
              if (kcslope < 0) then
                    ;si voix masculine
                    if (ksexef == 1) then
                    aamp2i = aamptmp2*aenvg*kenv*(3+(1.1*((400-if0)/300)))
                    aamp3i = aamptmp3*aenvg*kenv*(3+(1.1*((400-if0)/300)))
                    aamp4i = aamptmp4*aenvg*kenv*(3+(1.1*((400-if0)/300)))
                    aamp5i = aamptmp5*aenvg*kenv*(3+(1.1*((400-if0)/300)))
                    else
                    ;si voix féminine
                    aamp2i = aamptmp2*aenvg*kenv*(.8+1.05*((1000-if0)/1250))
                    aamp3i = aamptmp3*aenvg*kenv*(.8+1.05*((1000-if0)/1250))
                    aamp4i = aamptmp3*aenvg*kenv*(.8+1.05*((1000-if0)/1250))
                    aamp5i = aamptmp5*aenvg*kenv*(.8+1.05*((1000-if0)/1250))
                    endif
              else
                    ;si krslope est positif, il sert d'index pour la mise à
l'échelle
                    aamp2i = aamptmp2*krslope
                    aamp3i = aamptmp3*krslope
                    aamp4i = aamptmp4*krslope
                    aamp5i = aamptmp5*krslope
              endif
        ;seul les quatre derniers formants sont concerné par l'ajustement
```

```
        kenvar1=kenv ;conservation des effets de kenv sur le premier formant
        ;conservation des effets de aenvg sur le premier formant
        aengar1=aenvg
        ;annulation des effets de kenv sur les quatre derniers formants
        kenv=1
        ;annulation des effts de kenv sur les quatre derniers formants
        aenvg=1
endif

;Creation des 5 formants
ar1 fof aamp1i*kenv*aenvg*kenvar1*aengar1,
af0i+(if0*amod*kenv_vib)+kfond_var, afre1i, koct, kwid1i, kris, kdur, kdec,
iolaps, ifna, ifnb, itotdur, iphs, ifmode, iskip
ar2 fof aamp2i*kenv*aenvg, af0i+(if0*amod*kenv_vib)+kfond_var, afre2i,
koct, kwid2i, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, iphs, ifmode,
iskip
ar3 fof aamp3i*kenv*aenvg, af0i+(if0*amod*kenv_vib)+kfond_var, afre3i,
koct, kwid3i, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, iphs, ifmode,
iskip
ar4 fof aamp4i*kenv*aenvg, af0i+(if0*amod*kenv_vib)+kfond_var, afre4i,
koct, kwid4i, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, iphs, ifmode,
iskip
ar5 fof aamp5i*kenv*aenvg, af0i+(if0*amod*kenv_vib)+kfond_var, afre5i,
koct, kwid5i, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, iphs, ifmode,
iskip

asort = ar1 + ar2 + ar3 + ar4 + ar5 ;addition des 5 formants
out asort

endin
```

ANNEXE n°10 : traitement des informations par la fonction *vibra* de la bibliothèque *chant-lib*

La fonction *vibra* permet de créer une liste avec les différents paramètres du vibrato et les résultats des fonctions *jitter* et *envelope-vib*. Cette liste peut ensuite être envoyée à la fonction *chant* qui ajoutera son contenu à la liste finale *resform* (cf. (III)-(B)-(c)).

Afin de comprendre au mieux comment la fonction *vibra* traite les informations qui lui sont envoyées, il est nécessaire d'en étudier sa construction :

Tout d'abord, le nom de la fonction est déclaré ainsi que ses douze variables.

```
(om::defmethod! vibra ((vibamp number) (vala1 number)
                       (vala2 number) (tvala1 number)
                       (tvala2 number) (vibfreq  number)
                       (valf1 number) (valf2 number)
                       (tvalf1 number) (tvalf2 number)
                       &optional (jitter list) (env_vib list))
  :icon 234
  :indoc '("vibamp" "vala1" "vala2" "tvala1" "tvala2" "vibfreq" "valf1"
  "valf2" "tvalf1" "tvalf2" "jitter" "enveloppe vibrato")
  :initvals '(0.05 0.01 0.01 5000 5000 5 0.01 0.01 1000 1000 '(1) '(1))
```

La fonction *vibra* teste si la fonction *jitter* est connectée en se renseignant sur la taille de la liste associée à l'entrée *jitter*. Si celle-ci est égale à un, cela signifie que la valeur par défaut est utilisée `(1)` et que la fonction *jitter* n'est par conséquent pas connectée. Dans ce cas, les emplacements de la liste réservés aux paramètres de *jitter* sont remplis avec des zéros ce qui permet à CSOUND de savoir si la fonction *jitter* est connectée. Dans le cas inverse, les valeurs contenues dans la liste retournée par *jitter* sont extraites et copiées dans la liste créée par la fonction *vibra* : *vibamp*.

Un autre test du même type est effectué pour savoir si la fonction *envelope-vib* est connectée.

```
(cond
((= (length jitter) '1) (progn (if (= (length env_vib) 2)
                        (list vibamp vala1 vala2 tvala1
                              tvala2 vibfreq valf1 valf2
                              tvalf1 tvalf2 0 0 0 0 0 0
                              (nth 0 env_vib) (nth 1 env_vib))
                        (list vibamp vala1 vala2 tvala1
                              tvala2 vibfreq valf1 valf2
                              tvalf1 tvalf2 0 0 0 0 0 0 0 0))))
((> (length jitter) '1) (progn (if (= (length env_vib) 2)
                        (list vibamp vala1 vala2 tvala1
                              tvala2 vibfreq valf1 valf2 tvalf1 tvalf2
                              (nth 0 jitter) (nth 1 jitter) (nth 2 jitter)
                              (nth 3 jitter) (nth 4 jitter)
                              (nth 5 jitter) (nth 0 env_vib)
                              (nth 1 env_vib))
                        (list vibamp vala1 vala2 tvala1
                              tvala2 vibfreq valf1 valf2 tvalf1 tvalf2
                              (nth 0 jitter) (nth 1 jitter) (nth 2 jitter)
                              (nth 3 jitter) (nth 4 jitter)
                              (nth 5 jitter) 0 0))))))
```

Pour finir, il faut préciser que les valeurs contenues dans la liste retournée par la fonction *vibra* sont directement intégrées dans la liste *resform* de la fonction *chant* :

```
(nth 0 vibra)
(nth 1 vibra)
(nth n vibra)
[…]
```

Si *vibra* n'est pas connecté à la fonction *chant*, celle-ci utilisera les valeurs définies par défaut : une liste de zéro. Ceci permettra à CSOUND de savoir si la fonction *vibra* est connectée ou non à *chant*.

ANNEXE n°11 : création du vibrato dans l'orchestre CSOUND « chant.orc » à partir des informations transmises par la fonction *vibra* de la bibliothèque *chant-lib*

Le vibrato est créé dans CSOUND à partir des informations transmises par la fonction *vibra* à la fonction *chant* de la manière suivante :

CSOUND teste d'abord si la fonction *vibra* est connectée à *chant*. Si se n'est pas le cas, aucun vibrato n'est appliqué à la fondamentale.

```
if (p6=0 && p7=0 && p8=0 && p9=0 && p10=0 && p11=0 && p12=0 && p13=0 &&
p14=0 && p15=0) then
      amod = 0
else
```

Les deux jitters pour les variations aléatoires de l'amplitude sont créés à l'aide de l'*opcode* `randh`. Celui-ci permet de générer des valeurs comprises entre zéro et un nombre donné de façon aléatoire à une certaine fréquence. Son premier argument correspond à la valeur maximale pouvant être tirée au sort et son deuxième à la fréquence des tirages.[199]

Les nombres tirés aléatoirement doivent être centrés pour pouvoir agir dans le domaine des entiers relatifs.

Afin de ne pas avoir de discontinuités dans le vibrato, chaque valeur produite aléatoirement par la fonction `randh` est interpolée linéairement avec la suivante. Cette opération est effectuée grâce l'*opcode* `lineto` qui permet d'interpoler les valeurs qui lui sont fournies en un temps donné. Ainsi, le premier argument de `randh` correspond aux valeurs à interpoler (l'interpolation se fait à chaque nouvelle valeur avec la précédente) et son deuxième la durée d'interpolation en secondes qui est ici égale à $1/tvala_n$.[200]

```
      kvala1 = p7
      ktvala1 = p9
      kvala2 = p8
      ktvala2 = p10

      krand_amp1 randh kvala1*1000, ktvala1
      kavar_amp1 = 1+((krand_amp1/1000) - (kvala1/2))
      kavar_amp_interp1 lineto kavar_amp1, 1/ktvala1

      krand_amp2 randh kvala2*1000, ktvala2
      kavar_amp2 = 1+((krand_amp2/1000) - (kvala2/2))
      kavar_amp_interp2 lineto kavar_amp2, 1/ktvala2
```

---

199 VERCOE, Barry, « RANDH », *The canonical CSOUND Reference Manual, Version 5.09, éd. sous la direction de Barry Vercoe*, Cambridge : MIT, 2005, p. 1495-1496, disponible en ligne : http://www.csound.com/manual/ (en ligne le 04/09/2010).

200 VERCOE, Barry, *op. cit.*, « LINETO », p. 1015.

De la même manière, deux jitters introduisent des variations aléatoires à la fréquence du vibrato.

```
krand_freq1 randh kvalf1*1000, ktvalf1
kavar_freq1 = 1+((krand_freq1/1000) - (kvalf1/2))
kavar_freq_interp1 lineto kavar_freq1, 1/ktvalf1

krand_freq2 randh kvalf2*1000, ktvalf2
kavar_freq2 = 1+((krand_freq2/1000) - (kvalf2/2))
kavar_freq_interp2 lineto kavar_freq2, 1/ktvalf2
```

L'enveloppe de l'amplitude du vibrato est créée avec l'*opcode* `linen` qui permet de produire une enveloppe du type attaque – maintien – chute. Dans un premier temps, CSOUND teste si la fonction *envelope-vib* est connectée. Les valeurs produites par `linen` seront multipliées à celle du vibrato, elles seront donc comprises entre 0 et 1. Pour cette raison, la valeur passée dans le premier argument de `linen` est un, puisqu'elle correspond à la valeur maximale de l'amplitude. Les trois autres arguments contrôlent dans l'ordre : la durée de l'attaque, la durée totale de l'enveloppe et la durée de la chute.[201] Enfin, il est important de noter que la durée totale de l'enveloppe est réduite de 20 ms.

```
iatt_vib = p47
idurn_vib = p3-0.02
idec_vib = p48

if (iatt_vib != 0 && idec_vib != 0) then
    kenv_vib linen 1, iatt_vib, idurn_vib, idec_vib
else
    kenv_vib = 1
endif

kvalf1 = p12
ktvalf1 = p14
kvalf2 = p13
ktvalf2 = p15
```

Le vibrato est créé avec un oscillateur produisant un signal sinusoïdal (GEN10) multiplié à la fréquence f_0 de la fondamentale. Son amplitude *vibamp* est perturbée par un coefficient variant de façon aléatoire dont la valeur est donnée par la moyenne des valeurs des deux jitter décrits précédemment. Le même procédé est appliqué à la fréquence *vibfreq* du vibrato.

```
kvibamp = p6
kvibfreq = p11

amod    oscili    kvibamp*((kavar_amp_interp1+kavar_amp_interp2)/2),
kvibfreq*((kavar_freq_interp1+kavar_freq_interp2)/2), 1
endif
```

Le coefficient de modulation produit par les opérations décrites précédemment est associé à la variable *amod*. Il agit sur la fréquence de la fondamentale de la façon suivante : `if0 +`

201 VERCOE, Barry, *op. cit*, « LINEN », p. 1012-1013.

`(if0*amod*kenv_vib)`. Enfin, il est important de remarquer que l'enveloppe de l'amplitude du vibrato est utilisée lors de cette étape.

ANNEXE n°12 : création du jitter dans CSOUND dans l'orchestre « chant.orc »

Le vibrato est créé dans CSOUND à partir des informations transmises par la fonction *jitter* à la fonction *chant* de la manière suivante :

CSOUND teste dans un premier temps si la fonction *jitter* est connectée à *chant*. Si ce n'est pas le cas, aucune variation aléatoire n'est appliquée à la fondamentale.

```
if (p16=0 && p17=0 && p18=0 && p19=0 && p20=0 && p21=0) then
kfond_var = 0
```

Les valeurs aléatoires qui permettront la modulation de la fréquence de la fondamentale sont produites grâce à un ensemble de trois jitters. Chaque jitter est créé avec l'*opcode* `jitter` de CSOUND. Celui-ci permet de fournir les mêmes résultats qu'une combinaison des objets `randh` et `lineto` comme celle utilisée précédemment. Son premier argument correspond à l'amplitude de la modulation. Ici, les valeurs de *jitt1*, *jitt2* et *jitt3* agissent comme des proportions de la fréquence f_0. Le deuxième et le troisième argument permettent de déterminer la fréquence à laquelle chaque valeur aléatoire est produite.[202]

Une moyenne des résultats des trois jitters à un instant *t* est faite et permet de déterminer l'index de modulation de la fréquence f_0 de la fondamentale.

```
else
        kjitt1 = p16
        kjitt2 = p17
        kjitt3 = p18
        kjittf1 = p19
        kjittf2 = p20
        kjittf3 = p21

        kfond_var1 jitter kjitt1*if0, kjittf1, kjittf1
        kfond_var2 jitter kjitt2*if0, kjittf2, kjittf2
        kfond_var3 jitter kjitt3*if0, kjittf3, kjittf3
        kfond_var = (kfond_var1 + kfond_var2 + kfond_var3)/3
endif
```

Le jitter final produit lors des étapes décrites précédemment est directement additionné à la fréquence de la fondamentale au moment de la synthèse. On peut voir en orange dans la ligne de code suivante le récapitulatif des opérations de modulation de la fréquence f_0 :

```
arn fof aampni*kenv*aenvg, aint+(if0*amod*kenv_vib)+kfond_var, afreni,
koct, kwidni, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, iphs, ifmode,
iskip
```

202 VERCOE, Barry, *op. cit.*, « JITTER », p. 968-969.

ANNEXE n°13 : fonctionnement de la fonction *envelope2* de la bibliothèque *chant-lib*

Le mode de fonctionnement de *envelope2* s'inspire grandement de celui des fonctions *pargen57* et *table* de la bibliothèque *om2csound*. En effet, dans un premier temps les variables *y-min* et *y-max* sont déclarées. Elles serviront à mettre l'échelle de CSOUND les valeurs issues du BPF.

```
(let* ((y-min '0) (y-max '1) (ndec '3) res
```

Les différentes coordonnées *x* et *y* des points dessinés sur le BPF sont extraites sous forme de liste dans les variables *Ly* et *Lx*.

```
(Ly (copy-list (om::x-points bpf0)))
(Lx (copy-list (om::y-points bpf0))))
```

Une liste indiquant que la table de fonction à créer est la troisième de la future partition CSOUND, qu'elle reste active durant toute la performance, qu'elle a une précision de *pnts* points et qu'elle utilise GEN07[203] est créée. Elle est ensuite additionnée aux différents paramètres décrivant la forme de l'enveloppe dont les valeurs sont calculées par la fonction *paramxy* ainsi qu'à la liste des amplitudes de chaque note.

```
(push (append (list 'f '3 '0 pnts '7)
              (paramxy (om::om-round (om-scale/max Ly pnts))
              (om::om-round (om-scale Lx y-min y-max) ndec)))
      res)
(push amps res)
(reverse res))
```

La liste retournée par le fonction *envelope2* et envoyée à l'objet *chant* est donc de la forme : ((*tables de fonction*) (*liste des amplitudes de chaque note*)).

203 VERCOE, Barry, *op. cit.*, « GEN07 », p. 2197-2198.

ANNEXE n°14 : traitement par la fonction *chant* des informations retournées par les fonctions *envelope* et *envelope2* dans la bibliothèque *chant-lib*

Dans un premier temps, la fonction *chant* teste laquelle des deux fonctions pour le contrôle de l'enveloppe est connectée sur son entrée numéro quatre *envelope*. Si *envelope2* est connectée (cas où la liste fournie contient deux éléments) :

- la liste des amplitudes de chaque note est sauvegardée dans la variable *amps*
- la liste contenant la table de fonction issue du BPF est enregistrées dans la variable *fenv*
- les variables associées à la fonction *envelope* deviennent toutes égales à zéro
- la variable *envgen* devient égale à un, celle-ci permet d'indiquer à CSOUND que la fonction *envelope2* est utilisée.

Si *envelope* est connectée :

- la liste des amplitudes de chaque note est sauvegardée dans la variable *amps*
- une table de fonction par défaut est enregistrée dans la variable *fenv*
- les valeurs relatives à la durée de l'attaque, à la durée de la chute et à la durée de l'interpolation sont enregistrées respectivement dans les variables *att*, *rel* et *interp*
- la variable *envgen* devient égale à zéro afin d'indiquer à CSOUND que *envelope* est connectée.

```
(if (= (length envelope) 2)
       (setf amps (nth 1 envelope) fenv (nth 0 envelope)
             att 0 rel 0 interp 0 envgen 1)
       (setf amps (nth 3 envelope) fenv '(f 3 0 8192 10 1)
             att (nth 0 envelope) rel (nth 1 envelope)
             interp (nth 2 envelope) envgen 0))
```

ANNEXE n°15 : création d'enveloppes d'amplitude dans l'orchestre CSOUND « chant.orc »

Les enveloppes d'amplitude sont appliquées dans CSOUND à l'aide d'index mettant à l'échelle l'amplitude de chaque note ou d'une phrase musicale entière. Ils sont directement multipliés aux listes d'amplitudes et leur valeur varie entre zéro et un où un correspond donc à une amplitude maximum et zéro à un silence.

Dans un premier temps, des variables sont assignées à la durée de la note et aux différents paramètres de la fonction *envelope*. De la même manière que pour l'enveloppe du vibrato, la durée totale de l'enveloppe est ici réduite de 20ms afin de supprimer la perturbation du signal qui survient lors de l'arrêt d'un générateur de FOFs. Ce problème est inhérent à l'*opcode* `fof`.

Le premier test dans CSOUND vérifie si l'interpolation est activée. Si c'est le cas, les paramètres issus de la fonction *envelope* sont utilisés pour définir la forme de l'enveloppe d'amplitude de la phrase musicale. Si ce n'est pas le cas, CSOUND effectue une autre série de tests afin de savoir laquelle des fonctions *envelope* et *envelope1* est utilisée pour définir la forme de l'enveloppe de chaque note. On peut donc voir ici que seul la fonction *envelope* permet le contrôle de l'enveloppe d'amplitude de la phrase musicale dans son intégralité dans le cas ou l'interpolation linéaire est utilisée.

```
idurn = p3-0.02
iatt = p22
idec = p23
if (iinterp=0) then
```

Lorsque l'interpolation linéaire est désactivée, aucune enveloppe n'est appliquée à la phrase musicale dans sa totalité. Ainsi la variable *aenvg* qui est associée à l'enveloppe globale devient égale à un (cf. formule finale de modification de l'amplitude à la fin de cette annexe).

Si *envelope2* est connectée, la table de fonction *f3* est lue grâce à l'*opcode* `oscil1`. Ses différents arguments sont dans l'ordre[204] :

- la durée en seconde du délai avant que la table de fonction soit lue (ici zéro)
- l'amplitude maximum du signal produit (ici un)
- la durée en seconde pour lire la table de fonction (ici *idurn*)
- le numéro de la table de fonction utilisée.

Le résultat de la lecture est assigné à la variable *kenv* qui est donc utilisée pour contrôler l'enveloppe d'amplitude de chaque note.

```
   aenvg = 1
   if (p46=1) then
```

---

204 VERCOE, Barry, *op. cit.*, « OSCIL1 », p. 1234.

```
      kenv oscil1 0, 1, idurn, 3
   else
```

Si *envelope* est connectée, l'enveloppe d'amplitude de chaque note est calculée grâce à l'*opcode* `linen`[205].

```
      kenv linen 1, iatt, idurn, idec
   endif
else
```

Lorsque l'interpolation linéaire est activée, une enveloppe globale est appliquée à la phrase musicale en fonction des paramètres donnés à la fonction *envelope*. Cette enveloppe est créée à l'aide de l'*opcode* `linseg` et d'une variable globale *gicnt* dont les valeurs sont tenues d'un événement CSOUND à un autre.

De la même manière que précédemment pour *aenvg*, la variable *kenv* devient égale à un afin qu'elle n'est plus aucune incidence sur l'amplitude des son produit. La variable *gicnt* est initialisée à zéro au début de l'orchestre « chant.orc ». Ainsi, lors de la lecture de la première note, l'*opcode* `linseg`[206] est utilisé pour tracer deux segments successifs : un entre zéro et un en *iatt* secondes et un autre entre un et un en (*idurn* – *iatt*) secondes. Cette opération permet de définir le début de l'enveloppe globale de la phrase.

La variable *gicnt* est ensuite incrémentée de un à chaque fois qu'une note est jouée dans CSOUND.

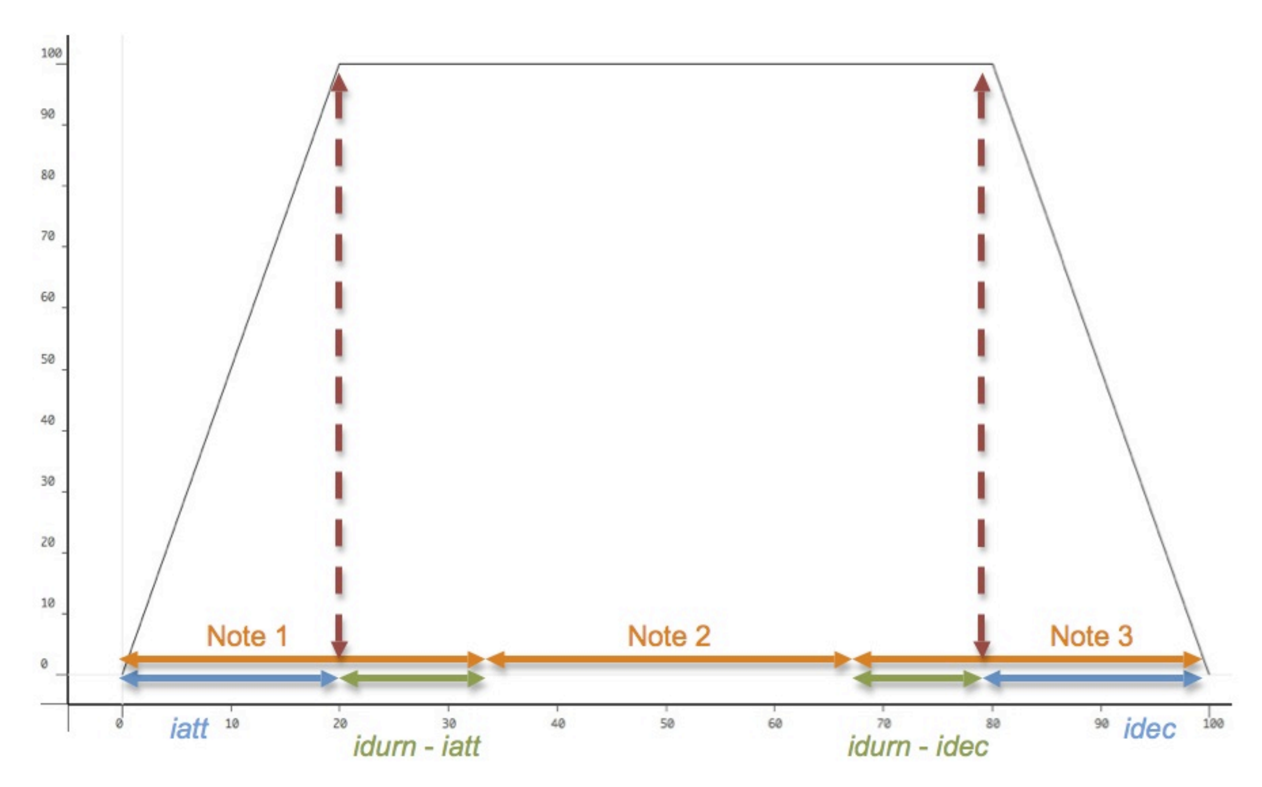
Lorsque la dernière note est jouée (*knotes* – 1) où *knotes* est le nombre total de notes dans la phrase musicale, l'*opcode* `linseg` est utilisé pour tracer un premier segment entre un et un en (*idurn* – *idec*) secondes puis un deuxième entre un et zéro en *idec* secondes. Il est aussi important de préciser que la variable *aenvg* est maintenue à un entre la première et la dernière note.

```
   knotes = p25
   kenv = 1
   if (gicnt == 0) then
       aenvg linseg 0, iatt, 1, (idurn - iatt), 1
   elseif (gicnt == (knotes - 1)) then
       aenvg linseg 1, (idurn - idec), 1, idec, 0
   else
       aenvg = 1
   endif
   gicnt = gicnt + 1

endif
```

205 VERCOE, Barry, *op. cit.*, « LINEN », p. 1012-1013.

206 VERCOE, Barry, *op. cit.*, « LINSEG », p. 1018-1020.



Création d’une enveloppe d’amplitude pour une phrase musicale complète dans CSOUND

Enfin, l’enveloppe d’amplitude de chaque générateur de formant est modulée de la façon suivante : *amplitude d’un générateur de formant * kenv * aenvg*.

ANNEXE n°16 : interpolation linéaire de la fondamentale, des amplitudes, des fréquences et des largeurs de bandes dans l'orchestre « chant.orc » pour CSOUND

L'interpolation linéaire est un principe fondamental dans la synthèse de la voix chantée. Cette technique permet non seulement d'introduire des glissandos entre les différentes notes d'une phrase musicale, mais aussi de passer d'une voyelle à une autre de façon naturelle (cf. (II)-(C)-(c)). Ainsi, l'ensemble des paramètres contrôlant les générateurs de FOFs (fréquence de la fondamentale, amplitude, fréquences et largeurs de bandes des formants) doivent êtres interpolés à chaque fois qu'une nouvelle valeur leur est soumise.

Cette opération est menée à bien dans CSOUND grâce à l'*opcode* `linseg`[207] et une collection de variables globales dont les valeurs sont maintenues d'un événement CSOUND à un autre.

Tout d'abord, CSOUND teste si l'interpolation est activée. Si c'est le cas la variable *iskip* devient égale à un. Celle-ci indique alors aux générateurs de FOFs que leur phase d'initialisation doit être sautée ce qui est primordial pour introduire des glissandos entre chaque note. Si l'interpolation est désactivée, *iskip* devient égale à zéro.

```
if (iinterp==0) then
      iskip = 0
else
      iskip = 1
endif
```

Le test est effectué une nouvelle fois mais avec une condition supplémentaire : la variable globale *gilast_tst* doit avoir une valeur supérieur à zéro. En effet, cette dernière permet d'empêcher l'interpolation lors de la première note. Pour ceci, elle est initialisée avec une valeur de zéro. Une fois que la première note d'une phrase musicale est jouée, *gilast_tst* devient égale à 1 autorisant alors l'interpolation.

```
if (gilast_tst>0 && iinterp!=0) then
```

Si l'interpolation est autorisée, CSOUND assigne la durée de l'interpolation à la variable *iduri*. L'interpolation est ensuite menée à bien grâce l'*opcode* `linseg` qui trace un segment entre la valeur précédente et la nouvelle valeur d'un des paramètres de la synthèse en un temps donnée ici défini par *iduri*. Les valeurs des différents paramètres sont sauvegardées d'une note à une autre dans des variables globales qui sont mise à jour à la fin de l'interpolation.

```
      iduri = iinterp
```

207 VERCOE, Barry, *op. cit.*, « LINSEG », p. 1018-1020.

Les fréquences f_0 de la fondamentale sont interpolées.

```
af0i linseg gilast, iduri, p4
```

Les fréquences des cinq formants sont interpolées.

```
afre1i linseg gifre1, iduri, ifre1
afre2i linseg gifre2, iduri, ifre2
afre3i linseg gifre3, iduri, ifre3
afre4i linseg gifre4, iduri, ifre4
afre5i linseg gifre5, iduri, ifre5
```

Les largeurs de bandes des cinq formants sont interpolées.

```
kwid1i linseg giwid1, iduri, iwid1
kwid2i linseg giwid2, iduri, iwid2
kwid3i linseg giwid3, iduri, iwid3
kwid4i linseg giwid4, iduri, iwid4
kwid5i linseg giwid5, iduri, iwid5
```

Les amplitudes des cinq formants sont interpolées.

```
aamp1i linseg giamp1, iduri, iamp1
aamp2i linseg giamp2, iduri, iamp2
aamp3i linseg giamp3, iduri, iamp3
aamp4i linseg giamp4, iduri, iamp4
aamp5i linseg giamp5, iduri, iamp5
```

Les variables globales pour l'interpolation sont mises à jour.

```
      gilast = p4
      gifre1 = ifre1
      gifre2 = ifre2
      gifre3 = ifre3
      gifre4 = ifre4
      gifre5 = ifre5

      giwid1 = iwid1
      giwid2 = iwid2
      giwid3 = iwid3
      giwid4 = iwid4
      giwid5 = iwid5

      giamp1 = iamp1
      giamp2 = iamp2
      giamp3 = iamp3
      giamp4 = iamp4
      giamp5 = iamp5
else
      gilast = p4
```

Les valeurs des différents paramètres de la première note jouée sont sauvegardées pour être réutilisées lors de la première interpolation entre la fin de la première note et le début de la deuxième.

```
gifre1 = ifre1
gifre2 = ifre2
gifre3 = ifre3
gifre4 = ifre4
gifre5 = ifre5

giwid1 = iwid1
giwid2 = iwid2
giwid3 = iwid3
giwid4 = iwid4
giwid5 = iwid5
```

```
        giamp2 = iamp2
        giamp3 = iamp3
        giamp4 = iamp4
        giamp5 = iamp5
```

Dans la mesure où les variables issues de l'interpolation sont directement utilisées par les générateurs FOFs (cf. fin du code) leur valeur devient égale à celle des valeurs non interpolées dans le cas où l'interpolation est désactivée.

```
        af0i = p4

        afre1i = ifre1
        afre2i = ifre2
        afre3i = ifre3
        afre4i = ifre4
        afre5i = ifre5

        kwid1i = iwid1
        kwid2i = iwid2
        kwid3i = iwid3
        kwid4i = iwid4
        kwid5i = iwid5

        aamp2i = iamp2
        aamp3i = iamp3
        aamp4i = iamp4
        aamp5i = iamp5
        gilast_tst = 1
endif
```

Les valeurs issues de l'interpolation sont finalement directement utilisées dans l'étape de synthèse :

```
arn fof aampni*kenv*aenvg, aint+(if0*amod*kenv_vib)+kfond_var, afreni,
koct, kwidni, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, iphs, ifmode,
iskip
```

ANNEXE n°17 : fonctionnement des fonctions *ffreq, ampf* et *bdwth* de la bibliothèque *chant-lib*

Les listes de valeurs fournies font partie intégrante de chaque fonction. Les valeurs qui y sont utilisées sont celles disponibles dans le manuel de CSOUND[208]. Il est possible des les consulter dans l'annexe n°24.

Dans la mesure où les trois fonctions dont il est ici question sont construites de la même façon, seul le code LISP de *ffreq* vat être ici présenté.

Dans un premier temps, le nom de la fonction est déclaré ainsi que ses différents arguments et leurs valeurs par défaut.

```
(om::defmethod! ffreq ((sexe t) (txt list))
  :icon 176
  :indoc '("sexe" "voyelles")
  :initvals '('soprano '(a))
```

Des listes de fréquences de formants sont associées à des variables représentatives d'un cas de figure particulier (un type de voix avec une certaine voyelle).

```
  (let* ((res)
            (txtl (length txt))
            ;frequences des formants de basse
            ;pour chaque voyelle
            (afb '(600 1040 2250 2450 2750))
            (efb '(400 1620 2400 2800 3100))
            (ifb '(250 1750 2600 3050 3340))
            (ofb '(400 750 2400 2600 2900))
            (ufb '(350 600 2400 2675 2950))

            ;frequences des formants de tenor
            ;pour chaque voyelle
            (aft '(650 1080 2650 2900 3250))
            (eft '(400 1700 2600 3200 3580))
            (ift '(290 1870 2800 3250 3540))
            (oft '(400 800 2600 2800 3000))
            (uft '(350 600 2700 2900 3300))

            ;frequences des formants de contretenor
            ;pour chaque voyelle
            (afct '(660 1120 2750 3000 3350))
            (efct '(440 1800 2700 3000 3300))
            (ifct '(270 1850 2900 3350 3590))
            (ofct '(430 820 2700 3000 3300))
            (ufct '(370 630 2750 3000 3400))

            ;frequences des formants d'alto
            ;pour chaque voyelle
            (afa '(800 1150 2800 3500 4950))
            (efa '(400 1600 2700 3300 4950))
            (ifa '(350 1700 2700 3700 4950))
            (ofa '(450 800 2830 3500 4950))
            (ufa '(325 700 2530 3500 4950))
```

208 VERCOE, Barry, *op. cit.*, « Appendix D. Formant Values », p. 2350-2354.

```
             ;frequences des formants de soprano
             ;pour chaque voyelle
             (afs '(800 1150 2900 3900 4950))
             (efs '(350 2000 2800 3600 4950))
             (ifs '(270 2140 2950 3900 4950))
             (ofs '(450 800 2830 3800 4950))
             (ufs '(325 700 2700 3800 4950)))
```

On teste quel type de voix est utilisé. La liste des voyelles est ensuite lue et on associe alors une liste de fréquences à chacune d'entre elles. Par exemple, dans le cas où deux « a » sont chantés par une soprano, la liste retournée par la fonction *ffreq* sera `((800 1150 2900 3900 4950) (800 1150 2900 3900 4950))`. Enfin, si un mauvais type de voix est entré ou si une faute d'orthographe a été commise, un message d'erreur est retourné dans le listener.

```
(case sexe
(basse (dotimes (n txtl res)
                (case (nth n txt)
                      (a (push afb res))
                      (e (push efb res))
                      (i (push ifb res))
                      (o (push ofb res))
                      (u (push ufb res)))))
(tenor (dotimes (n txtl res)
                (case (nth n txt)
                      (a (push aft res))
                      (e (push eft res))
                      (i (push ift res))
                      (o (push oft res))
                      (u (push uft res)))))
(contretenor (dotimes (n txtl res)
             (case (nth n txt)
                   (a (push afct res))
                   (e (push efct res))
                   (i (push ifct res))
                   (o (push ofct res))
                   (u (push ufct res)))))
(alto (dotimes (n txtl res)
               (case (nth n txt)
                     (a (push afa res))
                     (e (push efa res))
                     (i (push ifa res))
                     (o (push ofa res))
                     (u (push ufa res)))))
(soprano (dotimes (n txtl res)
                  (case (nth n txt)
                        (a (push afs res))
                        (e (push efs res))
                        (i (push ifs res))
                        (o (push ofs res))
                        (u (push ufs res)))))

(otherwise (om-beep-msg "Entrez un type de voix correct
(basse/tenor/contretenor/alto/soprano)!")))

(reverse res)))
```

ANNEXE n°18 : traitement des informations retournées par *ffreq* et *bdwth* par la fonction *chant* dans la bibliothèque *chant-lib*

Dans le cas des fonctions *ffreq* et *bdwth*, la fonction *chant* teste si ces dernières sont connectées. Si c'est le cas, leurs listes de paramètres sont transmises à CSOUND qui les utilisera pour la synthèse. Si elles ne sont pas connectées, des valeurs significatives sont envoyées à CSOUND (une liste de zéro) qui effectuera la synthèse avec des paramètres par défauts.

```
(if (< (length (nth 0 ffreq)) 5)
    (setf freq1 0 freq2 0 freq3 0 freq4 0 freq5 0)
    (setf freq1 (nth 0 (nth n ffreq)) freq2 (nth 1 (nth n ffreq))
          freq3 (nth 2 (nth n ffreq)) freq4 (nth 3 (nth n ffreq))
          freq5 (nth 4 (nth n ffreq))))

(if (< (length (nth 0 bdwth)) 5)
    (setf bdw1 0 bdw2 0 bdw3 0 bdw4 0 bdw5 0)
    (setf bdw1 (nth 0 (nth n bdwth)) bdw2 (nth 1 (nth n bdwth))
          bdw3 (nth 2 (nth n bdwth)) bdw4 (nth 3 (nth n bdwth))
          bdw5 (nth 4 (nth n bdwth))))
```

ANNEXE n°19 : traitement des largeurs, fréquences et amplitudes des formants dans l'orchestre CSOUND « chant.orc »

L'orchestre CSOUND « chant.orc » vérifie si les fonctions *ffreq*, *ampf* et *bdwth* sont connectées. Cette opération est menée à bien en contrôlant le contenu des différents p-fields relatifs aux paramètres des fonctions citées précédemment. Si l'ensemble des paramètres de l'une de ces fonctions est égal à zéro, alors CSOUND considère que la fonction n'est pas connectée et des valeurs par défaut (une soprano chantant la voyelle *a*) sont utilisées. Dans le cas inverse, les valeurs contenues dans les différents p-fields sont assignées à des variables qui seront ensuite utilisées lors du processus de synthèse.

```
if (p30==0 && p31==0 && p32==0 && p33==0 && p34==0) then
      ifre1 = 800
      ifre2 = 1150
      ifre3 = 2900
      ifre4 = 3900
      ifre5 = 4950
else
      ifre1 = p30
      ifre2 = p31
      ifre3 = p32
      ifre4 = p33
      ifre5 = p34
endif

if (p35 == 0 && p36 == 0 && p37 == 0 && p38 == 0 && p39 == 0) then
      iwid1 = 80
      iwid2 = 90
      iwid3 = 120
      iwid4 = 130
      iwid5 = 140
else
      iwid1 = p35
      iwid2 = p36
      iwid3 = p37
      iwid4 = p38
      iwid5 = p39
endif
```

Dans le cas de la fonction *ampf*, les amplitudes de formants en décibels sont traduites en dBfs avec un facteur de 0dbfs = 32767 grâce à la fonction `ampdb()` afin de les rendre compatibles avec les *opcodes* `fof`.

```
if (p26 == 0 && p27 == 0 && p28 == 0 && p29 == 0)
      iamp1 = ampdb(p5)
      iamp2 = ampdb(p5 - 6)
      iamp3 = ampdb(p5 - 32)
      iamp4 = ampdb(p5 - 20)
      iamp5 = ampdb(p5 - 50)
else
      iamp1 = ampdb(p5)
      iamp2 = ampdb(p5 + p26)
      iamp3 = ampdb(p5 + p27)
      iamp4 = ampdb(p5 + p28)
```

```
        iamp5 = ampdb(p5 + p29)
    endif
```

ANNEXE n°20 : traitement des informations envoyées à la fonction *amp-cor* de la bibliothèque *chant-lib*

Comme cela a été expliqué précédemment, la fonction *amp-cor* est la seule des règles de chant à ne pas être implémentée directement dans OpenMusic à cause des variations de l'amplitude de chaque formant en fonction de l'enveloppe. Elle se contente donc d'envoyer une liste de paramètres à CSOUND qui effectue les calculs relatifs à la règle.

Dans un premier temps, la fonction *amp-cor* et ses différentes entrées avec leurs valeurs par défaut sont déclarées.

```
(om::defmethod! amp-cor ((ampf list) (cslope number) (ajus list) (sexe t))
      :indoc '("Frequences des formants" "pente du spectre" "coefficients d
ajustement" "sexe")
      :initvals '('((-6 -32 -20 -50)) '1 '(1 1 0) 'soprano)
      :icon 141
```

Des variables locales sont déclarées. Le type de voix utilisé est testé et la fréquence médiane de sa tessiture est enregistrée dans la variable *f-mean*. Si le type de voix utilisée est une voix masculine, un est sauvegardé dans la variable *sexef*, si c'est une voix féminine, zéro. *cslope* ne peut pas être égal à zéro, ainsi, si *cslope*=0 cette valeur est remplacée par 0.000001.

L'ensemble des variables citées précédemment sont mises dans une liste qui contient également les trois coefficients d'ajustement *ajus*. Enfin, cette liste est placée au début de la liste contenant les listes d'amplitudes de formants passées dans l'entrée *ampf*. La liste finale est de la forme : ((*liste de paramètres de « amp-cor »)* (*liste*$_1$ *d'amplitudes de formants*) (*liste*$_n$ *d'amplitudes de formants*)).

```
      (let (f-mean sexef)
           (cond ((string= sexe 'soprano) (setf f-mean 523))
                 ((string= sexe 'alto) (setf f-mean 392))
                 ((string= sexe 'contretenor) (setf f-mean 370))
                 ((string= sexe 'tenor) (setf f-mean 261))
                 ((string= sexe 'basse) (setf f-mean 165))
           (if  (or  (string=  sexe  'contretenor)  (string=  sexe  'basse)
(string= sexe 'tenor))
           (setf sexef '1) (setf sexef '0))
           (when (zerop cslope) (setq cslope 0.000001))
           (push (list cslope (nth 0 ajus) (nth 1 ajus) (nth 2 ajus) f-mean
sexef) ampf)))
```

ANNEXE n°21 : traitement des informations envoyées par les fonction *amp-cor* et *ampf* à la fonction chant dans la bibliothèque *chant-lib*

Les fonctions *amp-cor* et *ampf* peuvent être connectées sur l'entrée *ampf* de la fonction *chant*. Dans la mesure où les listes fournies par ces deux fonctions concernant l'amplitude des formants sont différentes, il est nécessaire d'effectuer une série de tests.

Dans un premier temps, la fonction *chant* vérifie que l'une des deux fonctions est connectées (la première liste de la liste fournie doit contenir plus de quatre éléments). Si ce n'est pas le cas, des valeurs significatives sont envoyées à CSOUND qui utilisera des valeurs par défaut pour les amplitudes des formants et qui ne les ajustera pas. Si c'est le cas, d'autres tests sont effectués.

```
(dotimes (n notes resform)
(if (< (length (nth 0 ampf)) 4)
    (setf amp1 0 amp2 0 amp3 0 amp4 0 cslope 0 ajus1 0 ajus2 0 ajus3 0 f-
mean 0 sexef 0)
```

Si la fonction *amp-cor* est connectée (la première liste de la liste fournie compte six éléments), les paramètres pour le contrôle de la règle de correction de l'amplitude des formants sont extraits de la première liste et placés dans des variables tout comme les différentes amplitudes contenues dans les autres listes. Cette dernière opération est répétée pour chaque note et donc pour chaque liste issue de la fonction *amp-cor*.

```
    (if (= (length (nth 0 ampf)) 6)
        (setf cslope (nth 0 (nth 0 ampf)) ajus1 (nth 1 (nth 0 ampf))
              ajus2 (nth 2 (nth 0 ampf)) ajus3 (nth 3 (nth 0 ampf))
              f-mean (nth 4 (nth 0 ampf)) sexef (nth 5 (nth 0 ampf))
              amp1 (nth 0 (nth (+ n 1) ampf))
              amp2 (nth 1 (nth (+ n 1) ampf))
              amp3 (nth 2 (nth (+ n 1) ampf))
              amp4 (nth 3 (nth (+ n 1) ampf)))
```

Si la fonction *ampf* est connectée, des valeurs significatives sont envoyées à CSOUND qui n'utilisera pas la règle de correction de l'amplitude des formants. Les variables associées aux différentes amplitudes des formants sont remplies avec les résultats de *ampf*.

```
        (setf cslope 0 ajus1 0 ajus2 0 ajus3 0 f-mean 0 sexef 0
              amp1 (nth 0 (nth n ampf)) amp2 (nth 1 (nth n ampf))
              amp3 (nth 2 (nth n ampf)) amp4 (nth 3 (nth n ampf))))))
```

ANNEXE n°22 : correction des amplitudes des formants dans l'orchestre CSOUND « chant.orc » à partir des paramètres issus de la fonction *amp-cor* de la bibliothèque *chant-lib*

Conformément à l'algorithme présenté dans (II)-(C)-(f), une partie de l'orchestre « chant.orc » permet de corriger des amplitudes de formant en fonction de l'effort vocal.

Dans un premier temps, CSOUND contrôle si la fonction *amp-cor* a été connectée à la fonction *chant* dans OpenMusic. Si ce n'est pas le cas, la règle de correction de l'amplitude des formants n'est pas utilisée.

```
if (p40==0 && p41==0 && p42==0 && p43==0 && p44==0 && p45==0) then
```

Dans le cas inverse, les paramètres de la fonction sont d'abord extraits des différents p-fields qui leur ont été assignés dans des variables.

```
else
      kcslope = p40 ;pente du spectre
      kajus1 = p41 ;coefficient d'ajustement n°1
      kajus2 = p42 ;coefficient d'ajustement n°2
      kajus3 = p43 ;coefficient d'ajustement n°3
      kfmean = p44 ;fréquence médiane de la tessiture de la voix utilisée
      ksexef = p45 ;sexe
```

La règle présentée dans (II)-(C)-(f) est appliquée sur les amplitudes interpolées des formants *aamp$_n$i*. L'effort vocal est ici représenté par les variables *aenvg* et *kenv* contenant les indexes de modulation de l'amplitude pour l'enveloppe globale de la phrase musicale (*aenvg*) et pour l'enveloppe de chaque notes (*kenv*).

```
      atino = kenv*aenvg*(if0/kfmean)^kajus3
      krslope = kcslope*exp(kajus1*taninv(kajus2*log(if0/kfmean)))
      aamptmp2 = atino*aamp2i
      aamptmp3 = atino*aamp3i
      aamptmp4 = atino*aamp4i
      aamptmp5 = atino*aamp5i
      if (kcslope < 0) then

            if (ksexef == 1) then
                  aamp2i = aamptmp2*aenvg*kenv*(3+(1.1*((400-if0)/300)))
                  aamp3i = aamptmp3*aenvg*kenv*(3+(1.1*((400-if0)/300)))
                  aamp4i = aamptmp4*aenvg*kenv*(3+(1.1*((400-if0)/300)))
                  aamp5i = aamptmp5*aenvg*kenv*(3+(1.1*((400-if0)/300)))
            else
                  aamp2i = aamptmp2*aenvg*kenv*(.8+1.05*((1000-if0)/1250))
                  aamp3i = aamptmp3*aenvg*kenv*(.8+1.05*((1000-if0)/1250))
                  aamp4i = aamptmp3*aenvg*kenv*(.8+1.05*((1000-if0)/1250))
                  aamp5i = aamptmp5*aenvg*kenv*(.8+1.05*((1000-if0)/1250))
            endif
      else
            aamp2i = aamptmp2*krslope
            aamp3i = aamptmp3*krslope
            aamp4i = aamptmp4*krslope
            aamp5i = aamptmp5*krslope
            endif
```

```
      kenvar1=kenv
      aengar1=aenvg
      kenv=1
      aenvg=1
endif
```

ANNEXE n°23 : fonctionnement de la fonction *ampmid2db* de la bibliothèque *chant-lib*

La fonction *ampmid2db* permet d'implémenter le modèle pour le calcul de l'amplitude en décibels en fonction de la vélocité Midi présenté dans (III)-(B)-(j). Pour ceci, elle prend dans un premier temps en arguments une liste de vélocités Midi (*vel*, valeur par défaut : 1). La valeur de chaque élément de cette liste est testé afin de savoir quelle formule doit être utilisée en accord avec le modèle présenté précédemment. Si la valeur testée n'est pas comprise entre 0 et 127, un message d'erreur est retourné. Le résultat des conversions est stocké dans une variable locale dont le contenu est retourné à la fin des opérations :

```
(om::defmethod! ampmid2db ((vel list))
  :icon 178
  :indoc '("liste d amplitudes")
  :initvals '('(1))
  (let ((vell (length vel)) res)
      (dotimes (n ampl res)
            (cond
                  ((and (>= (nth n vel) 0) (< (nth n vel) 14))
                   (push (* (nth n vel) 0.14) res))
                  ((and (>= (nth n vel) 14) (< (nth n vel) 48))
                   (push (- (* (nth n vel) 0.45) 4.61) res))
                  ((and (>= (nth n vel) 48) (<= (nth n vel) 127))
                   (push (- (nth n vel) 31) res))
                  ((and (< (nth n vel) 0) (> (nth n vel) 127))
                   (om-beep-msg "L'argument entré n'est pas du type
midi"))))
      (reverse res)
  )
)
```

ANNEXE n°24 : paramètres des formants en fonction du type de voix et de la voyelle utilisée d'après de le manuel de CSOUND[209]

Soprano « a »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	800	1150	2900	3900	4950
Amp (dB)	0	-6	-32	-20	-50
Bw (Hz)	80	90	120	130	140

Soprano « e »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	350	2000	2800	3600	4950
Amp (dB)	0	-20	-15	-40	-56
Bw (Hz)	60	100	120	150	200

Soprano « i »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	270	2140	2950	3900	4950
Amp (dB)	0	-12	-26	-26	-44
Bw (Hz)	60	90	100	120	120

Soprano « o »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	450	800	2850	3800	4950
Amp (dB)	0	-11	-22	-22	-50
Bw (Hz)	40	80	100	120	120

209 VERCOE, Barry, *op. cit.*, « Appendix D. Formant Values », p. 2350-2354.

Soprano « u »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	325	700	2700	3800	4950
Amp (dB)	0	-16	-35	-40	-60
Bw (Hz)	50	60	170	180	200

Alto « a »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	800	1150	2800	3500	4950
Amp (dB)	0	-4	-20	-36	-60
Bw (Hz)	80	90	120	130	140

Alto « e »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	400	1600	2700	3300	4950
Amp (dB)	0	-24	-30	-35	-60
Bw (Hz)	60	80	120	150	200

Alto « i »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	350	1700	2700	3700	4950
Amp (dB)	0	-20	-30	-36	-60
Bw (Hz)	50	100	120	150	200

Alto « o »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	450	800	2830	3500	4950
Amp (dB)	0	-9	-16	-28	-55
Bw (Hz)	70	80	100	130	135

Alto « u »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	325	700	2530	3500	4950
Amp (dB)	0	-12	-30	-40	-64
Bw (Hz)	50	60	170	180	200

Contreténor « a »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	660	1120	2750	3000	3350
Amp (dB)	0	-6	-23	-24	-38
Bw (Hz)	80	90	120	130	140

Contreténor « e »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	440	1800	2700	3000	3300
Amp (dB)	0	-14	-18	-20	-20
Bw (Hz)	70	80	100	120	120

Contreténor « i »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	270	1850	2900	3350	3590
Amp (dB)	0	-24	-24	-36	-36
Bw (Hz)	40	90	100	120	120

Contreténor « o »

Valeurs	**f_1**	**f_2**	**f_3**	**f_4**	**f_5**
Freq (Hz)	430	820	2700	3000	3300
Amp (dB)	0	-10	-26	-22	-34
Bw (Hz)	40	80	100	120	120

Contreténor « u »

Valeurs	f_1	f_2	f_3	f_4	f_5
Freq (Hz)	370	630	2750	3000	3400
Amp (dB)	0	-20	-23	-30	-34
Bw (Hz)	40	60	100	120	120

Ténor « a »

Valeurs	f_1	f_2	f_3	f_4	f_5
Freq (Hz)	650	1080	2650	2900	3250
Amp (dB)	0	-6	-7	-8	-22
Bw (Hz)	80	90	120	130	140

Ténor « e »

Valeurs	f_1	f_2	f_3	f_4	f_5
Freq (Hz)	400	1700	2600	3200	3580
Amp (dB)	0	-14	-12	-14	-20
Bw (Hz)	70	80	100	120	120

Ténor « i »

Valeurs	f_1	f_2	f_3	f_4	f_5
Freq (Hz)	290	1870	2800	3250	3540
Amp (dB)	0	-15	-18	-20	-30
Bw (Hz)	40	90	100	120	120

Ténor « o »

Valeurs	f_1	f_2	f_3	f_4	f_5
Freq (Hz)	400	800	2600	2800	3000
Amp (dB)	0	-10	-12	-12	-26
Bw (Hz)	70	80	100	130	135

Ténor « u »

Valeurs	f_1	f_2	f_3	f_4	f_5
Freq (Hz)	350	600	2700	2900	3300
Amp (dB)	0	-20	-17	-14	-26
Bw (Hz)	40	60	100	120	120

Basse « a »

Valeurs	f_1	f_2	f_3	f_4	f_5
Freq (Hz)	600	1040	2250	2450	2750
Amp (dB)	0	-7	-9	-9	-20
Bw (Hz)	60	70	110	120	130

Basse « e »

Valeurs	f_1	f_2	f_3	f_4	f_5
Freq (Hz)	400	1620	2400	2800	3100
Amp (dB)	0	-12	-9	-12	-18
Bw (Hz)	40	80	100	120	120

Basse « i »

Valeurs	f_1	f_2	f_3	f_4	f_5
Freq (Hz)	250	1750	2600	3050	3340
Amp (dB)	0	-30	-16	-22	-28
Bw (Hz)	60	90	100	120	120

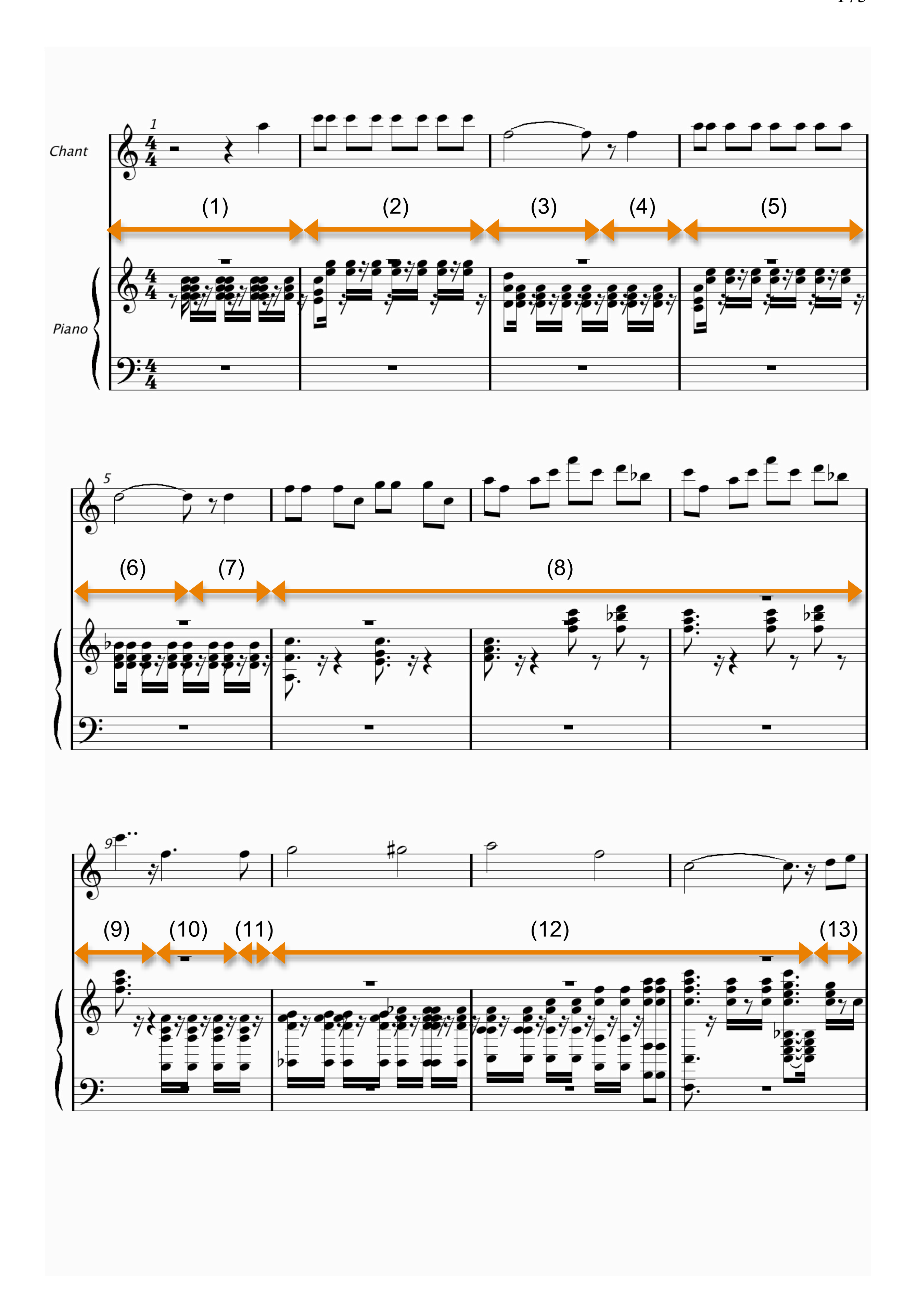
Basse « o »

Valeurs	f_1	f_2	f_3	f_4	f_5
Freq (Hz)	400	750	2400	2600	2900
Amp (dB)	0	-11	-21	-20	-40
Bw (Hz)	40	80	100	120	120
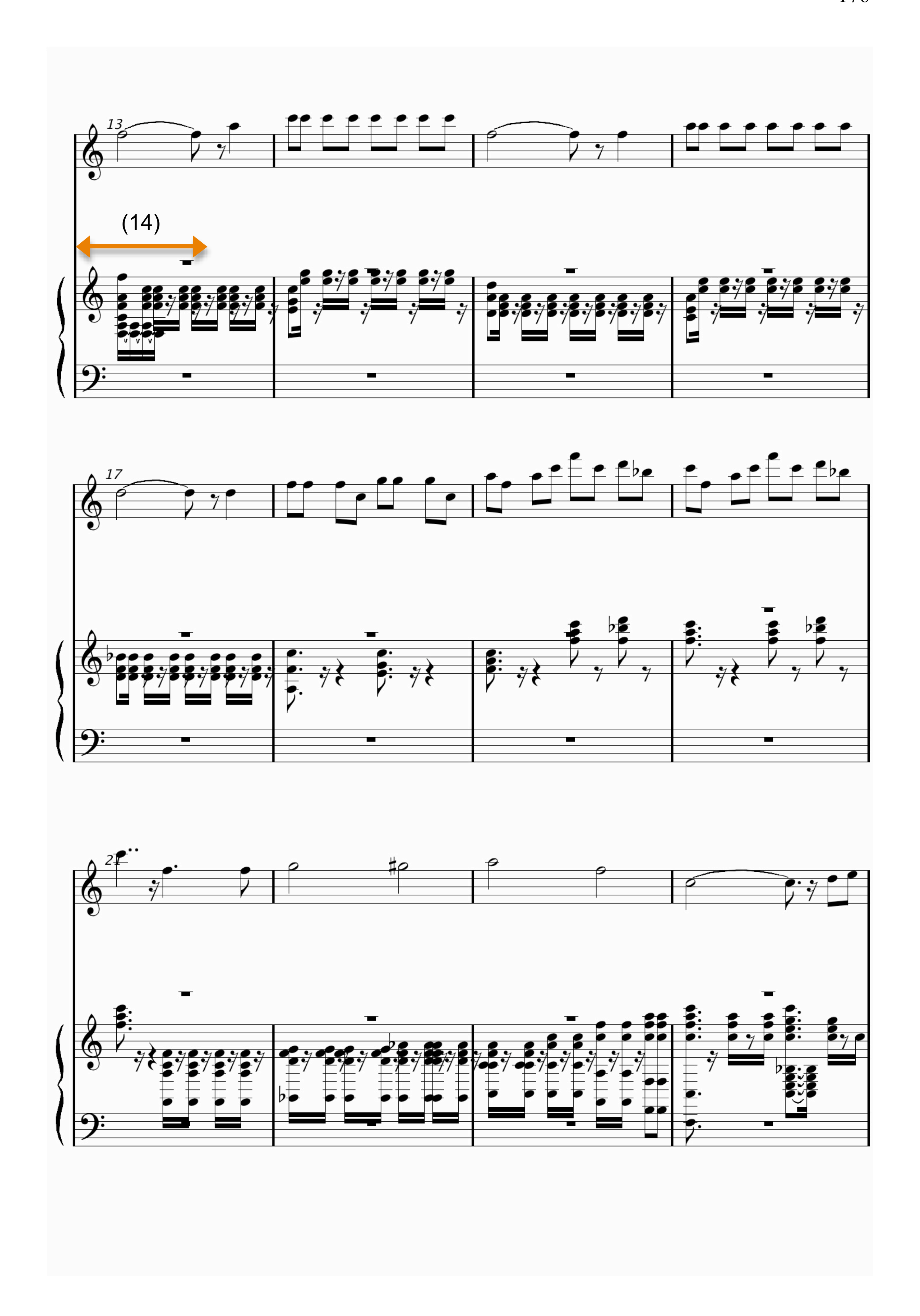
Basse « u »

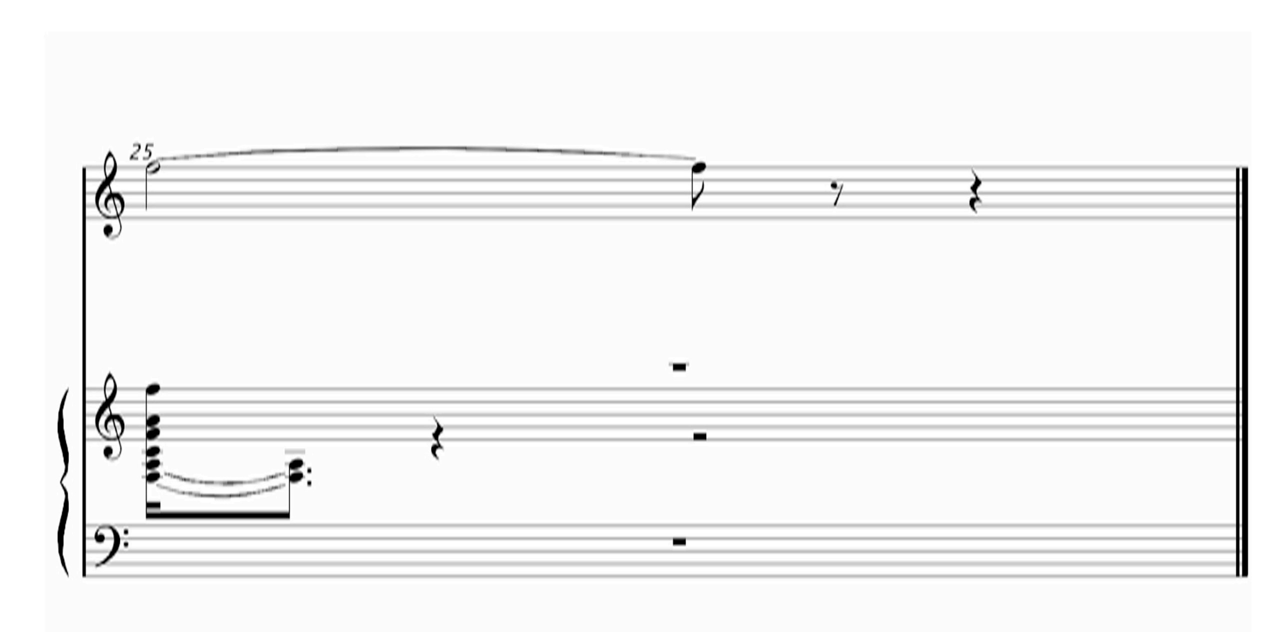
Valeurs	f_1	f_2	f_3	f_4	f_5
Freq (Hz)	350	600	2400	2675	2950
Amp (dB)	0	-20	-32	-28	-36
Bw (Hz)	40	80	100	120	120

ANNEXE n°25 : partition de *L'air de la reine de la nuit* de *La flute enchantée de Mozart* utilisée avec découpage des différentes section pour la synthèse et tableau des paramètres utilisés pour chaque section



Chant
Piano
1
5
9
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)


13
(14)
17
21



Extrait	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Durée de l'attaque des notes	0.1	0.04	0.1	0.1	0.04	0.1	0.1
Durée de la chute des notes	0.1	0.06	0.3	0.1	0.06	0.3	0.1
Durée de l'attaque du vibrato	0.1	X	0.1	0.1	X	0.1	0.1
Durée de la chute du vibrato	0.05	X	0.15	0.05	X	0.15	0.05
Amplitude du vibrato	0.1	0.01	0.1	0.1	0.01	0.1	0.1

Extrait	(8)	(9)	(10)	(11)	(12)	(13)	(14)
Durée de l'attaque des notes	0.04	0.1	0.1	0.04	0.1	0.04	0.1
Durée de la chute des notes	0.06	0.3	0.2	0.06	0.2	0.06	0.3
Durée de l'attaque du vibrato	X	0.1	0.1	X	0.1	X	0.1
Durée de la chute du vibrato	X	0 .15	0.1	X	0.1	X	0.15
Amplitude du vibrato	0.02	0.1	0.1	0.06	0.08	0.06	0.1

ANNEXE n° 26 : contenu du fichier CSOUND « fof-dem.csd »[210] (d'après l'exemple du manuel de CSOUND[211])

```
<CsInstruments>
; On utilise une fréquence d'échantillonnage de 44100 Hz
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1

kfund init 1 ;Fréquence de création de chaque impulsion, ici, une par
seconde
koct init 0 ;Index d'octaviation
kris init 0.003 ;Durée de l'attaque de l'impulsion : π/β
kdur init p3 ;Durée totale de l'impulsion
kdec init 0.007 ;Durée du decay de la FOF
iolaps = 60
ifna = 1 ;Utilisation d'une table de fonction de sinusoïde (GEN10) pour
l'impulsion
ifnb = 2 ;Utilisation d'une table de fonction de sigmoïde (GEN19) pour la
création de l'enveloppe
itotdur = p3 ;Durée de la performance, ici itotdur = kdur

;Caractéristiques du formant
k1amp = ampdb(89) ;amplitude max de chaque impulsion, ici 89 db
k1form init 2000 ;w0
k1band init 80 ;BW à -6 dB : α

;Générateur de FOF
a1 fof k1amp, kfund, k1form, koct, k1band, kris, \
kdur, kdec, iolaps, ifna, ifnb, itotdur

out a1
endin
</CsInstruments>

<CsScore>
f 1 0 4096 10 1 ;Table1 : Sinusoide
f 2 0 1024 19 0.5 0.5 270 0.5 ;Table2 : Sigmoïde
i 1 0 0.02
</CsScore>
```

---

210 Disponible sur le cd à cd/csound.

211 VERCOE, Barry, « FOF », *The canonical CSOUND Reference Manual, Version 5.09, éd. sous la direction de Barry Vercoe*, Cambridge : MIT, 2005, p. 750-751.

Bibliographie

Monographies – Articles

ALLESSANDRO, Christophe (d') ; RODET, Xavier, « Synthèse et analyse-synthèse par fonctions d'ondes formantiques », *Journal d'Acoustique*, n° 2 (1989), p. 163-169.

ARROABARREN, Ixone ; CARLOSENA, Alfonso ; RODET, Xavier, « On the Measurement of the Instantaneous Frequency and Amplitude of Partials in Vocal Vibrato », *Transaction on Audio, Speech and Language Processing*, XIV (2006), n° 4, p. 1413-1421.

ASSAYAG, Gérard ; AGON, Carlos, *OpenMusic omChroma: Paradigms for the high-level musical control of sonic processes using omChroma*, Paris : IRCAM, 2000.

ATAL, Bishnu, « Speech Analysis and Synthesis by Linear Prediction of the Speech Wave », *Journal of the Acoustical Society of America*, XLVII (1971), n° 65(A), p. 637-645.

BAISNEE, Pierre-François, *CHANT manual*, Document Ircam : Paris, 1985.

BOITE, René, *Traitement de la parole*, Lausanne : Presses polytechniques et universitaires romandes, 2000, p.16.

BONADA, Jordi ; CELMA, Oscar ; LOSCOS, Alex ; ORTOLA, Jaume ; SERRA, Xavier, *Singing Voice Synthesis Combining Excitation plus Resonance and Sinusoidal plus Residual Models : actes de l'International Computer Music Conference, La Havane, 2001*, http://mtg.upf.edu/files/publications/ (en ligne le 02/05/2010).

BONADA, Jordi ; SERRA, Xavier, « Synthesis of the Singing Voice by Performance Sampling and Spectral Models », *Signal Processing Magazine*, XXIV (2007), n° 2, p. 67-79.

BOSSIS, Bruno, *La voix et la machine : La vocalité artificielle dans la musique contemporaine*, Rennes : Presses Universitaire de Rennes, 2005.

BRESSON, Jean, *Sound Processing in OpenMusic : actes de la neuvième Conference on Digital Audio Effects (DAFx-06), Montréal, 18-20/09/2006*, Paris : IRCAM.

CHAMPION, Gaël, *Application du modèle additif « shape invariant » pour la transformation de la voix*, Mémoire de stage de DEA ATIAM, Université Paris VI, 2004.

CHARPENTIER, Francis, *Traitement de la parole par Analyse/synthèse de Fourier, application à la synthèse par diphones,* Thèse de doctorat (inédite), ENST de Paris, 1988.

CHARPENTIER, Francis ; MOULINES, Eric, « Pitch Synchronous Waveform Processing Techniques for Text-to-Speech Synthesis Using Diphones », *Speech Communication*, IX (1990), p. 453-467.

CHOWNING, John, « Frequency Modulation Synthesis of the Singing Voice », *Current Directions in Computer Music Research*, éd. sous la direction de Max Mathews et John Pierce, Cambridge : The MIT press, p. 57-64.

COOK, Perry-Raymond, *Identification of Control Parameters in a Articulatory Vocal Tract Model, with Applications to the Synthesis of Singing*, Thèse de doctorat (inédite), Université de Stanford, 1991.

COOK, Perry-Raymond, « Singing Voice Synthesis : History, Current Work, and Future Directions », *Computer Music Journal*, XX (1996), n°3, p. 38-46.

COOK, Perry-Raymond, « SPASM, a Real-Time Vocal Tract Physical Model Controller and Singer », *Computer Music Journal*, XVII (1993), n° 1, p. 30-44.

DEPALLE, Philipe ; GARCIA, Guillermo ; RODET, Xavier, *A Virtual Castrato : acte de l'International Computer Music Conference, Aarhus (Danemark), octobre* 1994, http://articles.ircam.fr/textes/Depalle94a/ (en ligne le 04/09/2010).

FLETCHER, Harvey ; MUNSON, William, « Loundness, its definition, measurement and calculation », *Journal of the Acoustical Society of America*, V (1933), n° 2, p. 82-108.

GAYOU, Evelyne, *John Chowning : portraits polychromes*, Paris : INA, 2005.

GUILBERT, Jean, *La parole : compréhension et synthèse par les ordinateurs*, Paris : Presses Universitaires de France, 1979.

HADDAD, Karim, *om2csound : bibliothèque de modules pour la génération de scores pour Csound (version 1)*, Paris : IRCAM, 1999.

HADDAD, Karim, *OpenMusic User's Manual*, Paris : IRCAM, 2003.

IOVINO, Francisco ; LAURSON, Mikael, *PatchWork PW-Chant reference*, Paris : IRCAM, 1996.

JANDER, Owen, « Singing », *Grove Music Online*, 2010, http://www.oxfordmusiconline.com/subscriber/article/grove/music/25869Singing (en ligne le 04/09/2010).

KELLY, John ; LOCHBAUM, Carol, *Speech synthesis : actes du Fourth International Congress on Acoustics, Copenhagen, septembre 1962*.

LEE, Matthew, *Acoustic Models for the Analysis and Synthesis of the Singing Voice*, Thèse de doctorat inédite, Georgia Institute of Technolgy, 2005.

LEMMETTY, Sami, *Review of Speech Synthesis Technology*, Mémoire de master, Helsinki University of Technology, 1999, disponible en ligne à l'adresse suivante : http://www.acoustics.hut.fi/publications/files/theses/lemmetty_mst/ (en ligne le 04/09/2010).

LIENARD, Jean-Sylvain, *Les processus de la communication parlée*, Paris : Masson, 1977.

LOIZILLON, Guillaume, *Diphone Studio: User Manual and Tutorial*, Paris : IRCAM, 1999.

MAC AULAY, Robert ; QUATIERI, Thomas, « Speech Analysis/Synthesis Based on a Sinusoidal Representation », *Transaction on Acoustics, Speech an Signal Processing*, XXXIV (1986), n° 4, p. 744-754.

MERON, Yoram, *High Quality Singing Synthesis using the Selection-based Synthesis Scheme*, Thèse de doctorat (inédite), Université de Tokyo, 1999.

MILLIKAN, Franck Rives, *Joseph Henry and the Telephone*, http://siarchives.si.edu/history/jhp/joseph23.htm (en ligne le 04/09/2010).

POTTIER, Laurent, « Le contrôle de la synthèse sonore par ordinateur », *Le Calcul de la Musique*, éd. sous la direction de Laurent Pottier, Saint-Etienne : Publication de l'Université de Saint-Etienne, 2009, p. 225-330.

POTTIER, Laurent, *Le contrôle de la synthèse sonore, le cas particulier du programme PatchWork*, Thèse de doctorat (inédite), Paris : EHESS, 2001.

PEETERS, Geoffroy, *Analyse et synthèse des sons musicaux par la méthode PSOLA : actes des cinquièmes journées de l'informatique musicale, La Londe-les-Maures, 5-7/05/1998*, Marseille : Publications de LMA, n° 148, 1998, p. G4-1 – G4-6.

PEETERS, Geoffroy ; RODET, Xavier, *SINOLA : A New Analysis/Synthesis Method using Spectrum Peak Shape Distortion, Phase and Reassigned Spectrum : actes de l'International Computer Music Conference (IMC), Pékin, 1999*, Pékin : Computer Music Association, 1999, p. 153-156.

PRESCOTT, George Bartlett, *Bell's Electric Speaking Telephone*, New-York : D. Appleton and company, 1884.

ROADS, Curtis, *The Computer Music Tutorial*, Cambridge : MIT Press, 1996.

RODET, Xavier, *Analyse du signal vocal dans sa représentation amplitude-temps : synthèse de la parole par règles*, Thèse de doctorat (inédite), Université Paris VI, 1977.

RODET, Xavier, *Synthesis of the Singing Voice : acte du colloque Benelux Workshop on Model Based Processing and Coding of Audio, Leuven (Belgique), 15/11/2002*, Paris : IRCAM, 2003.

RODET, Xavier, « Time-Domain Formant-Wave-Function Synthesis », *Computer Music Journal*, VIII (1984), n° 3, p. 9-14.

RODET, Xavier ; BARRIERE, Jean-Baptiste ; POTARD, Yves, *Rapport de recherche n°35 : Chant, de la synthèse de la voix à la synthèse en général*, Paris : IRCAM, 1985.

RODET, Xavier ; MANOURY, Philippe ; LEMOUTON, Serge ; PEETERS, Geoffroy ; SCHNELL, Norbet, *Synthesizing a choir in real-time using Pitch Synchronous Overlap Add (PSOLA) : actes du First Benelux Workshop on Model Based Processing and Coding of Audio, Bruxelles, 2002*, Paris : IRCAM, 2002.

RODET, Xavier ; PEETERS, Geoffroy, *SINOLA: A New Analysis/Synthesis Method using Spectrum Peak Shape Distortion, Phase and Reassigned Spectrum : actes de l'International Computer Music Conference, Pékin, 1999*, Paris : IRCAM, 1999.

SCEMAMA, Patrick, *K…, opéra en douze scènes de Philippe Manoury*, Paris : Opéra national de Paris, 2000.

SCHROEDER, Manfred R., « A Brief History of Synthetic Speech », *Speech Communication*, XIII (1993), n°1 et 2, p. 231-237.

SERRA, Xavier ; SMITH, Julius, « Spectral Modeling Synthesis: a Sound Analysis/Synthesis System Based on a Deterministic plus Stochastic Decomposition », *Computer Music Journal*, XIV (1990), n° 4, p. 12-24.

SMETS-SOLANES, Jean-Paul, *Un composant de synthèse formantique adapté à l'interaction multimédia : actes des 3èmes journées de l'informatique musicale, Île de Tatihou (France), 16-18/05/1996*, http://recherche.ircam.fr/equipes/repmus/jim96/actes/smets/qtfof.html (en ligne le 04/09/2010).

SMITH, Julius, *Musical Application of Digital Waveguides*, Stanford University Center for Computer Research in Music an Acoustics : Stanford Publication STAN-M-39, 1987.

SUNDBERG, Johan, *Synthesising Singing : actes de la quatrième Sound and Music Computing Conference, Lefkada (Greece), 11-13/07/2007*, http://www.smc-conference.org/smc07/ (en ligne le 04/09/2010).

SUNDBERG, Johan, *The Science of the Singing Voice*, Dekalb (Illinois) : Northern Illinois University Press, 1987.

VERCOE, Barry, « FOF », *The canonical Csound Reference Manual, Version 5.09*, éd. sous la direction de Barry Vercoe, Cambridge : MIT, 2005, disponible en ligne : http://www.csounds.com/manual/ (en ligne le 04/09/2010).

WINSTON, Patrick-Henry ; HORN, Berthold-Klaus-Paul, *LISP Third Edition*, Reading (Massachusetts) : Addison-Wesley, 1989.

Sites Internet

http://www.aamhl.org/dBchart.htm Page contenant le diagramme de Fletcher-Munson pour les correspondances entre amplitudes et nuances musicales (en ligne le 04/09/2010).

http://brahms.ircam.fr/works/work/6632/ Page contenant les notes de programme de *Chréode I* de Jean-Baptiste Barrière (en ligne le 04/09/2010).

https://ccrma.stanford.edu/~bilbao/master/node5.html Page du cours « *The Kelly-Lochbaum Digital Speech Synthesis Model* » de Stefan Bilbao au CCRMA de Stanford (en ligne le 04/09/2010).

http://common-lisp.net/ Site officiel de la communauté travaillant sur Common LISP (en ligne le 04/09/2010).

http://cours.musique.umontreal.ca/mus1321/Notes_de_cours/Csound_06_Voix.html Site du cours *analyse, synthèse et traitement de la voix* d'Olivier Belanger de l'université de Montréal (en ligne le 04/09/2010).

http://cslu.cse.ogi.edu/tts/research/sing/sing.html Page du programme de synthèse vocale « Lyricos » développé au Georgia Tech Music Department (en ligne le 04/09/2010).

http://www.csounds.com/ Site officiel du programme CSOUND (en ligne le 04/09/2010).

http://cycling74.com/products/Max/Mspjitter/ Site officiel commercial du programme Max/MSP (en ligne le 04/09/2010).

http://forumnet.ircam.fr/703.html Site officiel du programme Diphone Studio développé par l'IRCAM (en ligne le 04/09/2010).

http://mediatheque.ircam.fr/sites/voix/ Site Présentant les principaux outils d'analyse et de synthèse de la voix chantée avec leur utilisation dans quelques œuvres du répertoire de la musique contemporaine (en ligne le 04/09/2010).

http://ncvs.org/ncvs/tutorials/voiceprod/tutorial/filter.html Site internet du National Center for Voice and Speech (NCVS), page de tutoriaux sur la production de la voix (en ligne le 04/09/2010).

http://www.oxfordmusiconline.com/ Site du Grove Music Online (en ligne le 04/09/2010).

http://recherche.ircam.fr/equipes/analyse-synthese/home.html Site de l'équipe 'analyse-synthèse' de l'IRCAM (en ligne le 16/01/2010).

http://recherche.ircam.fr/equipes/analyse-synthese/reine.html Sur cette page, il est possible de télécharger le fichier audio de la synthèse de *L'air de la reine de la nuit* de *La Flûte enchantée* de Mozart (en ligne le 04/09/2010).

http://recherche.ircam.fr/equipes/repmus/OpenMusic/index.html Site officiel du programme OpenMusic (en ligne le 04/09/2010).

http://speech.bme.ogi.edu/tts/flinger/ Site du programme de synthèse vocale « Flinger » développé à l'université d'Edimbourg (en ligne le 04/09/2010).

http://tokyobling.wordpress.com/tag/music/ Blog de Tokyobling (en ligne le 04/09/2010).

http://www.vivos.fr/ Site du projet de recherche « Vivos » sur l'utilisation de voix de synthèse dans des applications multimédias (en ligne le 16/01/2010).

http://www.vocaloid.com/ Site officiel non commercial du programme « Vocaloid » développé par Yamaha (en ligne le 04/09/2010).

http://voce.cs.princeton.edu/ Site de Perry Cook sur le projet « voce » (en ligne le 04/09/2010).

http://weston.ruter.net/projects/ipa-chart/view/ Site officiel de l'API[212] (en ligne le 04/09/2010).

212 Alphabet Phonétique International.

Enregistrements

BARRIERE, Jean-Baptiste, « Chreode », *Computer Music Currents 4*, 1 CD Wergo WER 2024-50 (enreg. : 1983).

BROSCHI, Ricardo ; HANDEL, Georg-Friedrich ; HASSE, Johann-Adolf, *Farinelli, il castrato : bande originale du film*, Ewa Malas-Godlewska ; Derek Ragin ; Les talents lyrique, 1 CD Naïve V 5191 (enreg. : 1994).

CHOWNING, John, *Phoné*, 1 CD Wergo WER 2012-50 : 1988 (enreg. : 1981).

GRISEY, Gérard, *Les chants de l'amour, für 12 Stimmen und Tonband*, Schola Heidelberg, Enregistrement IRCAM non publié (enreg. : 28/11/2008).

HARVEY, Jonathan, *Mortuos Plango, Vivos Voco*, 1 CD Sargasso Records SCD 28029 : 1999.

Vidéos

Close your eyes, chanté par Kaito et Meiko, les voix virtuelles créées par Yamaha, vidéo disponible sur le cd dans le fichier « vocaloid.mp4 » à cd/vidéo et sur le site YouTube à l'adresse suivante : http://www.youtube.com/watch?v=cVU-tWu9LoY (en ligne le 04/09/2010).

Extrait du making-of du film *Il castrato*, vidéo disponible sur le cd dans le fichier « farinelli.mp4 » à cd/vidéo et sur le site YouTube à l'adresse suivante : http://www.youtube.com/watch?v=No-w5l00pho (en ligne le 04/09/2010).

Interview télévisée de Max Mathews sur la musique assistée par ordinateur, vidéo disponible sur le cd dans le fichier « Mathews-reine.m4v » à cd/vidéo et sur le site YouTube à l'adresse suivante : http://www.youtube.com/watch?v=_15ZQL82P4M (en ligne le 04/09/10).

Partitions

MANOURY, Philippe, *K : opéra pour 15 chanteurs électroniques et grand orchestre*, Partition d'orchestre, Paris : Durand, 2001.

Rencontres

BRESSON, Jean, IRCAM, Paris, 16/11/2009.

CHOWNING, John, *Interview Enregistrée*, Centre Interdisciplinaire d'Etude et de Recherche sur l'Expression Contemporaine (CIEREC), St Etienne, 09/11/2009, enregistrement disponible sur le CD à cd/audio/interview-chowning.m4a.

HADDAD, Karim, IRCAM, Paris, 19/11/2009.

Index des différents fichiers du CD-ROM

~/audio :

farinelli.aiff
FM.aif
interview-chowning.m4a
rein-IRCAM.aiff
transitions-voyelles.aif

~/chant-lib :

chant-lib.lisp
chant.orc
~/chant-lib/resources :
~/chant-lib/resources/icon/ :
Icones des différentes fonctions de la bibliothèque chant-lib
~/chant-lib/sources :
general.lisp
objets.lisp
regles.lisp
reglessc.lisp

~/csound :

fof-dem.aiff
fof-dem.csd
fof-sop-vib.aiff
fof-sop.aiff
fof-sop.csd

~/matlab :

chantbw.m
fof.m
fofd.fig
fofd.m
fofenv.fig
fofspec.fig
fofspec.m
sigmo.m

~/om :

ampf+ffreq+bdwth.omp
envelope.omp
envelope1+2.omp
exemple-complet.omp
exemple-patch.omp
exemple-reine.omp
ffreq+bdwth.omp
fof-chroma.omp
midi.omp
om2csound.omp
ons-dur-freq.omp
règles.omp

synthese-base.omp
test-dynamique.omp
vibra+jitter+envelope-vib.omp

~/PW :
voice-rules.Lisp

~/reine :
~/reine/cubasse :
Projet Cubase pour le montage des différentes sections de la synthèse de L'air de la reine de la nuit de La flute enchantée de Mozart
~/reine/découpage-audio :
Fichier audio des différentes sections synthétisées de L'air de la reine de la nuit de La flute enchantée de Mozart
~/reine/découpage-midi :
Fichiers Midi utilisés par les patchs dans ~/reine/découpage-audio pour la synthèse de L'air de la reine de la nuit de La flute enchantée de Mozart
~/reine/om :
Patchs OpenMusic pour la synthèse de L'air de la reine de la nuit de Mozart
reine-final.mid
reine-original.MID
reine-synth.aif

~/test-dynamique :
test-dynamique.aif
test-dynamique.mid
test-dynamique.omp
test-dynamique.xlsx
test-dynamique.maxpat

~/video :
farinelli.mp4
Mathews-reine.m4v
vocaloid.mp4

Index

Table des matières